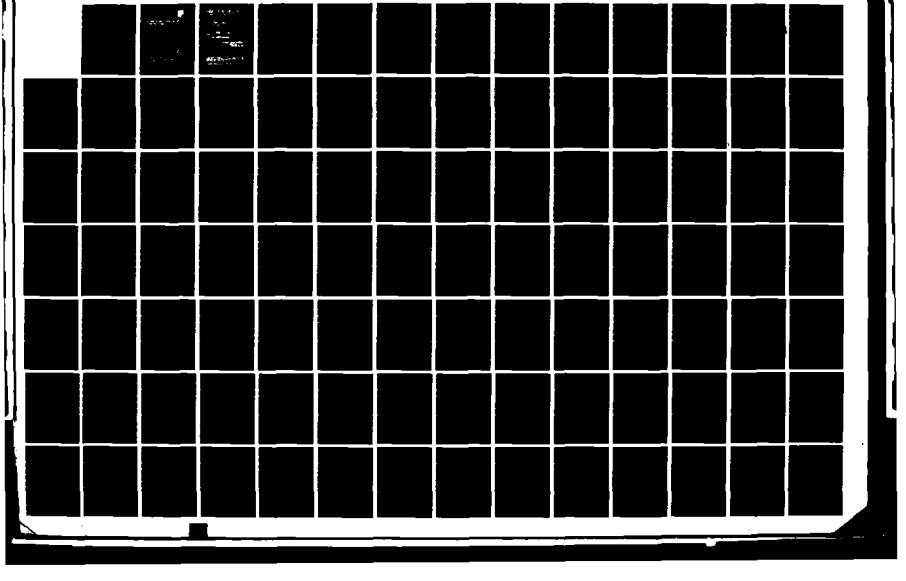
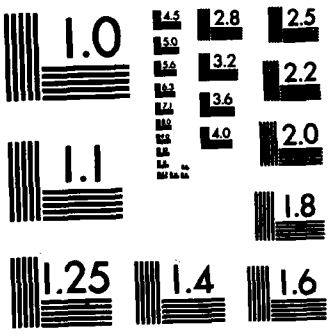
AD-A123 421 SOFTWARE RELIABILITY MODELLING AND ESTIMATION TECHNIQUES(U) SYRACUSE UNIV N Y DEPT OF INDUSTRIAL ENGINEERING AND OPERATIONS RESEARCH A L GOEL OCT 82 1/4

UNCLASSIFIED RADC-TR-82-263 F30602-78-C-0351 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

AD A123421

RADC-TR-82-263
Final Technical Report
October 1982

SOFTWARE RELIABILITY MODELLING AND ESTIMATION TECHNIQUES

Syracuse University

Amrit L. Goel

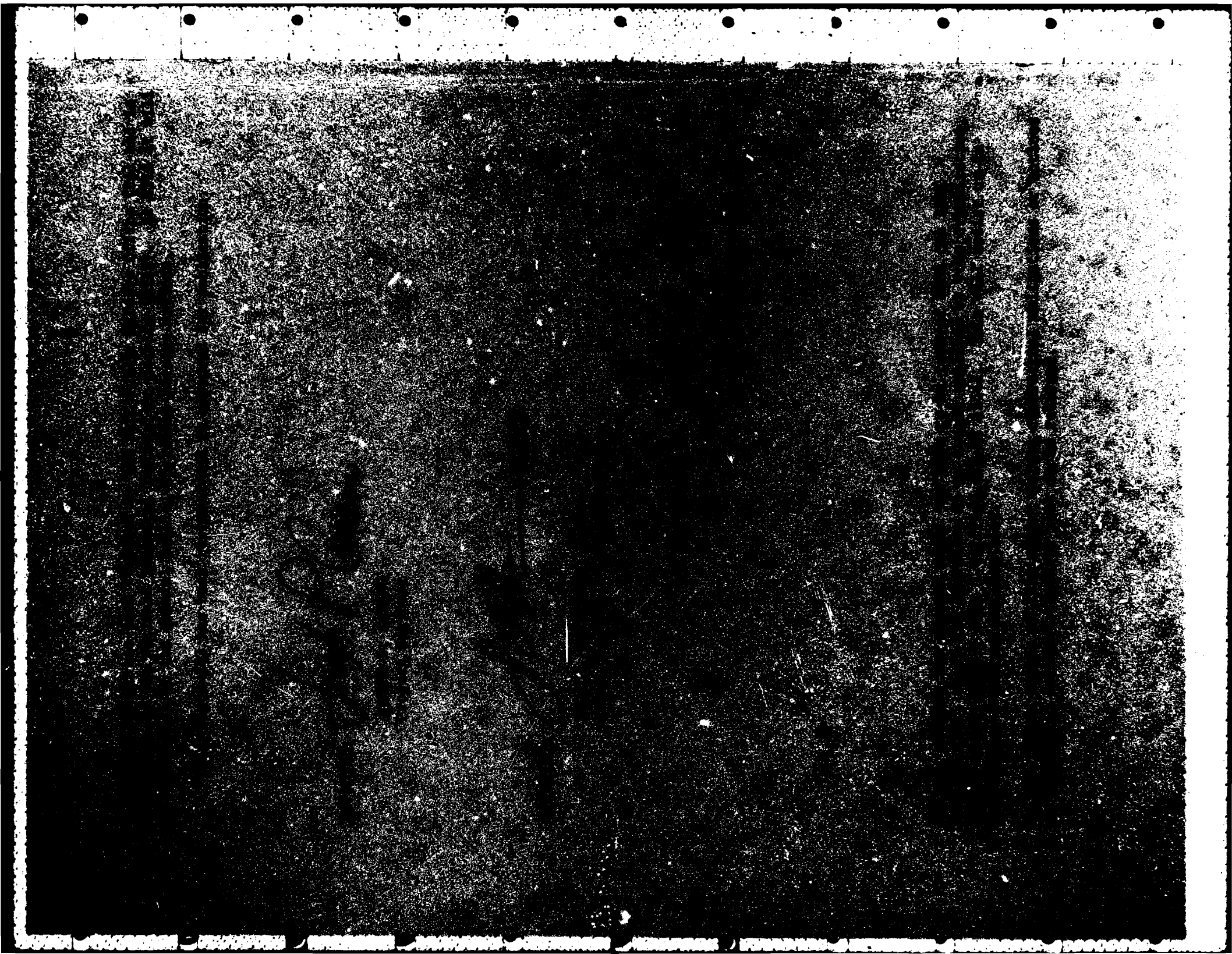
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
ELECTE
JAN 1 7 1983
S
A

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

83 01 17 050

DTIC FILE COPY



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-82-263	2. GOVT ACCESSION NO. AD-A123421	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE RELIABILITY MODELLING AND ESTIMATION TECHNIQUES		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report Oct 78 - Oct 81
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Amrit L. Goel		8. CONTRACT OR GRANT NUMBER(s) F30602-78-C-0351
9. PERFORMING ORGANIZATION NAME AND ADDRESS Syracuse University Department of Industrial Engineering Syracuse NY 13210		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55812015
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (COEE) Griffiss AFB NY 13441		12. REPORT DATE October 1982
		13. NUMBER OF PAGES 340
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Joseph P. Cavano (COEE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Reliability
Non-homogeneous Poisson Process
Software Release Time
Hardware-Software Reliability/Availability Models
Model Validation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents the results of the software reliability modelling and estimation research pursued under Contract F30602-78-C-0351 during the period October 1978 - October 1981.

Two new models of very general applicability are introduced and the necessary mathematical and practical details are developed in this report. A new methodology for determining when to stop testing and start using software is described and developed. (Cont'd on reverse)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Finally, a new model for analyzing the operational performance of a combined hardware-software system is reported even though it was not a part of the original research plan.

ACKNOWLEDGEMENT

Much of the work reported here was pursued with K. Okumoto, who was a research assistant at Syracuse University, and is currently employed by Bell Laboratories. I owe him special thanks for his diligence in the pursuit of this work. Hardware-software reliability modelling research was done with J. Soenjoto, who is currently employed by the Indonesian Government. His help with this work is greatly appreciated.

Messers Ashish Deb and Peter Valdes provided invaluable assistance with data analyses and programming, and I thank them for their work. I have also benefited greatly from discussions with Peter Valdes regarding the ideas about software reliability described in Chapter 1.

Finally, I would like to thank Messers Alan Sukert and Joseph Cavano of RADC for their many suggestions regarding the direction of this research work.

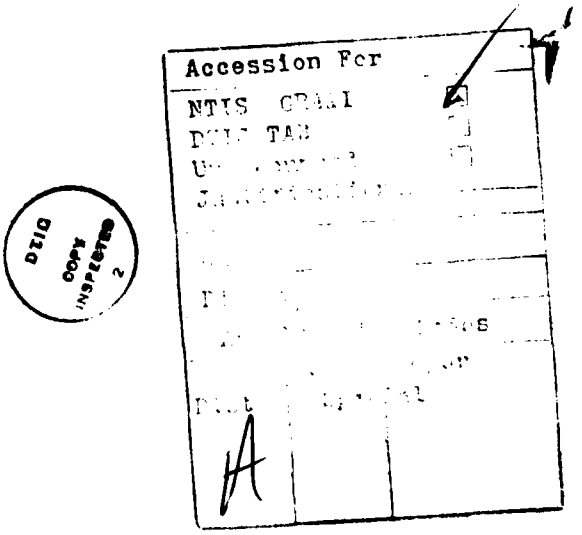


TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SECTION 1

INTRODUCTION AND OVERVIEW

An important quality attribute of a computer system is the degree to which it can be depended upon to perform its intended function in the specified environment.

Evaluation and prediction of this attribute has concerned computer designers and users from the early days of their evolution. Until the late sixties, the attention was almost solely on the performance of hardware aspects of the system. In the early seventies, software became the center of attention due to a continuing increase in the ratio of software to hardware costs, in both the production and the operational phases.

The performance of a software system is dependent on the tools and methodologies used during its development, and an important measure of performance is the nature and frequency of software errors.

This report is primarily concerned with the development of stochastic models for describing the software error occurrence phenomenon and determining software reliability. A description of software errors and their sources is given in Section 1.1 and an error classification scheme

is described in Section 1.2. The notion of software reliability is discussed in Section 1.3. Current suggested approaches for enhancing software reliability are given in Section 1.4. A description of software reliability models reported in the literature is given in Section 1.5.

A non-homogeneous Poisson process (NHPP) model based on an exponentially decaying error occurrence rate is developed in Section 2. Many useful software performance measures are developed and several software failure data sets are analyzed to show the applicability and usefulness of this model.

In Section 3, another NHPP model is proposed which can be used to model both the increasing and the decreasing failure rates during the software integration testing phase.

The problem of when to stop testing and start using software is discussed in Section 4. Various useful scenarios are considered and optimum release time policies are developed. The results are illustrated via numerical examples.

A related problem of modelling the total hardware-software system is addressed in Appendix A. This task

was not a part of the research work under the reference contract and the results are reported here as they are considered useful for people interested in software reliability modelling.

1.1 SOFTWARE ERRORS AND THEIR SOURCES

Software (also called program) is essentially an instrument for transforming a discrete set of inputs into a discrete set of outputs (see Figure 1.1). It comprises of a set of coded statements whose function may basically be one of the following:

1. Evaluate an expression and store the result in a temporary or permanent location.
2. Decide which statement to execute next.
3. Perform input/output operations.

Since, to a large extent software is produced by humans, the finished software product is often imperfect. It is imperfect in the sense that a discrepancy exists between what the software can do versus what the user or the computing environment wants it to do. The computing environment refers to the physical machine, operating system, compiler and translators, utilities, etc. These discrepancies are what we call <u>software errors</u> (see Figure 1.2). Basically, the software errors can be attributed to the following:

1. Ignorance of the user requirements;
2. Ignorance of the rules of the computing environment; and

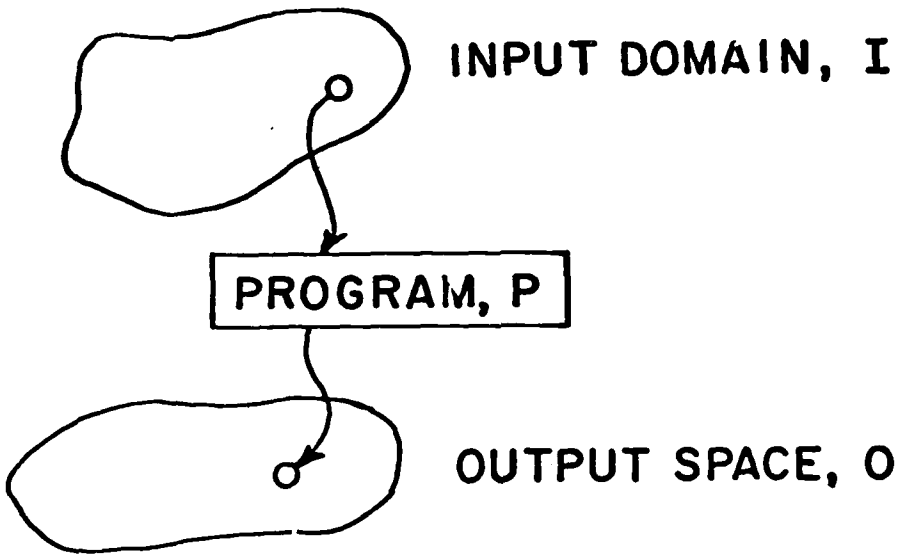


FIG. 1.1. FUNCTIONAL VIEW OF SOFTWARE

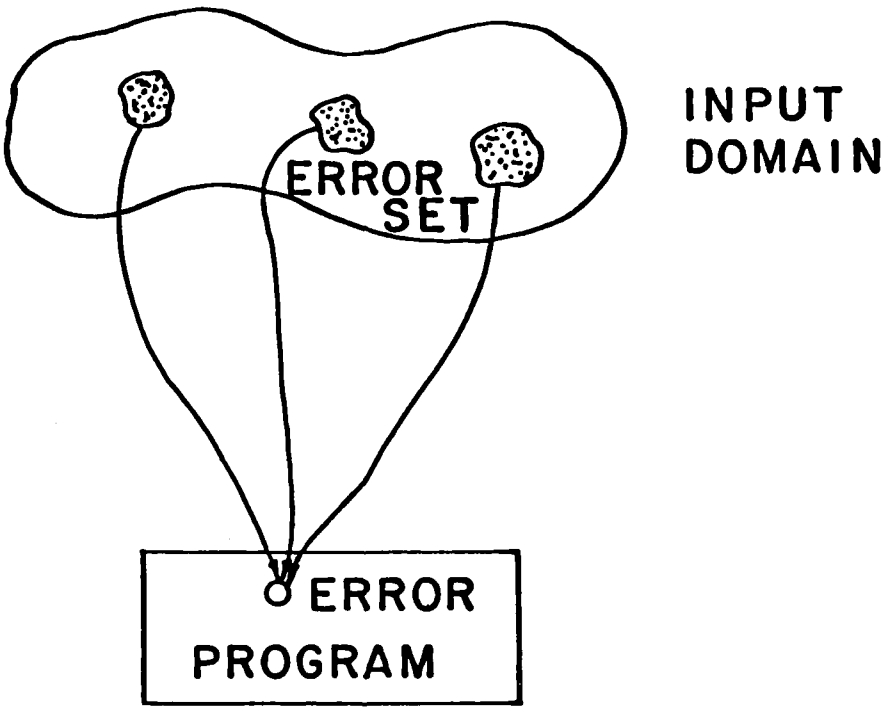


FIG. 1.2. SOFTWARE ERROR

3. Poor communication of software requirements between the user and the programmer or poor documentation of the software by the programmer.

The fact of the matter is even if we know that software contains errors, we may not know with certainty the exact identity of these errors.

Currently, there are two major paths one can follow to expose software errors:

1. Program proving, and
2. Program testing.

Program proving is more formal and mathematical while program testing is more practical and still remains to be heuristic in its approach. The approach in program proving is the construction of a finite sequence of logical statements ending in the statement (usually the output specification statement) to be proved. Each of the logical statements is an axiom or is a statement derived from earlier statements by the application of an inference rule. Program proving making use of inference rules is known as the Inductive Assertion Method. This method was mainly popularized by Floyd, Hoare, Dijkstra and recently Reynolds. Other work on program proving is the work on the Symbolic Execution Method. This method is the basis of some automatic program verifiers. Despite the formalism

and mathematical exactness of program proving, it is still an imperfect tool for verifying program correctness. Gerhart and Yelowitz [GER 76] showed several programs which were proven to be correct but still contained errors. The errors were due to failures in defining what exactly to prove and were not failures of the mechanics of the proof itself.

Program testing is the symbolic or physical execution of a set of test cases with the intent of exposing embedded errors (if any) in the program. Like program proving, program testing remains an imperfect tool for verifying program correctness. A given testing strategy is good for exposing certain kinds of errors but not all possible kinds of errors in a program. An advantage of testing is that it provides accurate information about a program's actual behavior in its actual computing environment; proving is limited to conclusions about the program's behavior in a postulated environment.

Neither proving nor testing can, in practice, guarantee complete confidence on the correctness of programs. Each has its pluses and minuses. They should not be viewed as competing tools. They are, in fact, complementary methods for decreasing the likelihood of program failure [GOO 77].

1.2 SOFTWARE ERROR CLASSIFICATION

A systematic study of software errors in a program requires knowing what specifically these errors are and knowing which tool(s) to use to expose particular types of software errors. Software errors can be grouped as syntax, semantic, runtime, specification and performance errors.

1.2.1 Syntax Errors

These errors are due to discrepancies between the program code and the syntax rules governing the parser or lexical analyzer of a program translator. These are the easiest errors to detect. They can be detected by visual inspection of the code or can be detected mechanically during the program compilation process. Experienced programmers rarely commit syntax errors.

1.2.2 Semantic Errors

These errors are due to discrepancies between the program code and what the semantic analyzer of the computing environment accepts. Among the popular kinds of semantic errors are typechecking errors and implementation restriction errors. Again, they may be detected by the semantic analyzer of a program translator or by visual inspection.

Syntax and semantic errors are detected during the compilation stage of a program. A program having syntax and/or semantic errors cannot be executed. Syntax and semantic errors are mainly due to the ignorance/negligence on the part of the programmer about the restrictions and limitations of the language (s)he is using.

1.2.3 Runtime Errors

As the name implies, runtime errors occur during the actural running of a program. They may be further classified into three categories:

Domain errors

A domain error occurs whenever the value of a program variable exceeds its declared range or exceeds the physical limits of the hardware representing the variable. The declared range of a variable is done implicitly or explicitly. FORTRAN, for example, assigns types to variables based on the variable name or based on a declaration statement. PASCAL requires all variables to be explicitly declared in a declaration statement. PASCAL has facilities to declare ranges by enumeration and/or subsets of numeric domains.

Some program translators produce runtime code for checking certain types of domain errors. Some have built-in recovery features for domain errors (e.g. PL/1, COBOL) and others (e.g. FORTRAN) simply abort execution upon the occurrence of a domain error. Certain compilers, like

PASCAL, automatically check for values outside a declared range.

Domain errors are a serious matter because

a) program execution is aborted, and/or

b) program results are incorrect.

Execution abortion may be fatal especially in real-time systems. Despite their seriousness, domain errors have never been formally and extensively studied in the literature. This is because detection of domain errors can be very difficult. They require exact specification of the ranges of the input variables. Also, the test values required to expose these errors may occur at the input domain's boundary or inside the input domain itself.

Computational errors

Computational errors, sometimes known as logic errors, result whenever the program results in an incorrect output. The incorrect output may be due to a wrong formula, an incorrect control flow, assignment to a wrong variable, incorrect parameter passing, etc.

It is not possible to generate runtime code to detect computational errors during program execution. This is because computational errors are really discrepancies between the program's output and the program's specifications.

Computational errors due to incorrect program constructs and statements may be detected by any of the structure dependent or structure independent testing techniques. However, none of these tools can guarantee total absence of these types of computational errors in a program. Computational errors due to missing program constructs and statements may be detected by any of the structure independent testing techniques. Again, none of these tools can guarantee total absence of computational errors due to missing paths.

Non-Termination errors

Non-termination error is simply the failure of a program to terminate in finite time without outside intervention. The most common cause of non-termination errors is when the program runs into an infinite loop. Non-termination can also occur if a set of concurrent programs falls into a dead lock.

Infinite loops are detected by simply executing each of the loops in a program. However, this strategy may not guarantee total absence of infinite loops. Some infinite loops may only occur if certain program variables achieve certain values. Program proving may also be used on certain programs to expose infinite loops. The problem of program non-termination in general is still an unsolved problem.

1.2.4 Specification Errors

Specification errors result whenever there exists a discrepancy between the statement of specifications and the statement of user requirements. A requirements error exists whenever there is a discrepancy between the statement of user requirements and the real user requirements.

Presently, detection of specification errors such as;

1. Incomplete specifications,
2. Inconsistent specifications, and
3. Ambiguous specifications,

remains an informal process. This is mainly due to the nonexistence of a specification language powerful enough to translate the user requirements into clear, complete and consistent terms.

A testing tool to detect specification errors is yet to be developed.

1.2.5 Performance Errors

Performance errors exist whenever a discrepancy exists between the actual performance (efficiency) of the programs and its desired or specified performance. Program performance may be measured in a number of ways:

1. Response time
2. Elapsed time
3. Memory space usage
4. Working set requirement, etc.

The actual measurement of the above measures of program performance can be a very difficult process. Program complexity theory tries to estimate bounds on the running time of certain program algorithms. Statistical analysis and simulation can also be employed to estimate the above performance variables. However, use of these tools can be very expensive and time consuming.

A performance testing tool that is economical (timewise and costwise) to use is yet to be developed.

The most expensive kind of software errors to eliminate are those which are not discovered until late in the software development, such as when the software becomes operational. These are known as persistent software errors. Glass [GLA81] reported that persistent software errors are mostly due to the failure of the problem solution (i.e. the program) to match the complexity of the problem to be solved (i.e. the user requirements). Examples of such errors are computational errors due to missing or insufficient predicates and failure to reset a variable to some baseline value after its use in a functional logic segment. The solution to this software problem is beyond the current state-of-the-art.

1.3 SOFTWARE RELIABILITY

There are a number of conflicting views as to what software reliability really is and how it should be quantified. The conflict arises because of the disagreement in the basic definition of the term "software reliability". Software reliability in the view of some people, especially the computer science purists, should be closely tied to the correctness of software. They argue that an incorrect software (i.e., a software still containing errors) is doomed to fail sooner or later and thus its reliability should be zero (0). Once the software has been freed of all errors, then its reliability becomes one (1). On the other hand, software reliability, as viewed by many engineers, statisticians, and practitioners, should be closely tied to the concept of "probabilistic reliability". These groups of people argue that many programs used in the real world are known to still contain errors and yet they are executed day after day without occurrences of failures. Software reliability, they believe, should be viewed as the probability that a software system will operate without a failure for a specified (mission) time.

One way to resolve this conflict is to look back at the original problem in the real world and ask ourselves the question: "Why do we need to know software reliability?"

The original real world problem, in very simple terms, is as follows:

Develop software that will satisfy the user's requirements in the most efficient (in both time and money sense) way possible.

The solution to this problem turns out to be very difficult basically because of the following facts:

1. Real world software is large and complex.
2. Users are not always 100 percent certain about their requirements.
3. Resources (time and money) allocated for software development are always limited.

Even if we know that we only need, say, 2000 test cases to run for exposure all possible embedded errors in a software, chances are that, in the real world, we may not have enough time and money to perform this exhaustive test. As more and more errors are uncovered by our testing or correctness verification process, the additional cost of exposing the other remaining errors rises very fast. Thus, beyond a point it is almost practically useless to continue testing to achieve 100 percent correctness. This explains the reason why most all software systems in public and private use still have embedded errors.

If we adopt the point of view of a computer science purist, then almost all software systems in use (including those that are accepted as very reliable and useful by their users) have zero reliability. Since everything now has zero reliability, the value or usefulness of the software reliability, concept is lost.

The reason why people introduced the concept of software reliability (or hardware reliability for that matter) is to have a useful measure that may help us in dealing with the original real world software (hardware) problem. This measure is useful in planning and controlling additional resources (time and money) for enhancing the reliability of a software. It is also a useful measure for giving the user confidence about the software quality.

Should we, then, adopt the hardware based concept of software reliability? One answer to this question at this point in time is yes, but with extreme care. We should be careful because there are inherent differences between software and hardware. Hardware exhibits mixtures of decreasing and increasing failure rates. The decreasing failure rate is due to the fact that as use time on the hardware system accumulates, more and more errors (most probably design errors) are encountered and fixed. The increasing failure rate is due primarily to hardware component

wearout. There is no such thing as wearout in software. It is true that software may become obsolete because of changes in the user and computing environment but once we modify software to reflect these changes, then we are no longer talking of the same software but an enhanced or modified version. Like hardware, software exhibits a decreasing failure rate as the usage time on the system accumulates and errors (due to design and coding) are fixed. Thus, a hardware-type approach to software reliability should be done only in appropriate environments.

Suppose we declare that the reliability of a given software is 0.95. What does this number exactly mean? Following the probabilistic point of view, this may mean any one of the following:

1. If we execute the software several times, 95 percent of the time it will give correct results.
2. We are 95 percent confident that the software will give correct results when executed.

The first interpretation is the so-called frequency interpretation and the second is the so-called subjective interpretation. Littlewood's contention [LIT80] is that in the absence of a "scientific" verifiable meaning for the number 0.95, the only reasonable interpretation is the subjective interpretation.

The only problem we see with this number is its possible inconsistency. A software may have been declared 95 percent reliable by the developer but may have a different perceived reliability by the user and probably a different percieved reliability by another user. A very simple example will illustrate this point. Suppose a software is composed of 100 modules. Because of practical considerations, the software developer stops testing after 90 modules. He then declares the reliability of the system as 90 percent. A user buys the system and happens to use in his particular application some modules (or maybe program paths) which have not been tested. As a result, 50 percent of the time, the user gets incorrect results. His perceived reliability of the system is therefore 50 percent. Another user might use a different mixture of untested modules (program paths) and might get a different number for the reliability measure. The basic question is: What is the true reliability of the software?

The only way to resolve this question, we feel, is to further qualify or condition our software reliability measure. Ultimately, what is more important is that the user gets his correct results from the software. Thus, that user should be more concerned about a reliability measure conditioned with respect to his requirements. The software developer should be concerned with a reliability measure conditioned with respect to the intended specifica-

tions of the system. We should remember that the purpose of the reliability measure is to help in planning and controlling the production of software and nothing more. We may pool all the users into one big user (for example, user of an operating system software) and come up with an average reliability measure. Still, this number may not match the developer's measured reliability. If we let R[S|r] mean the reliability of the software system S with respect to requirements r, then in general, we have:

$$R[S|\text{user requirements}] \neq R[S|\text{developer requirements}]$$

1.4 APPROACHES FOR ENHANCING SOFTWARE RELIABILITY

Consider the concept of software reliability based on the following definition [MYE76]:

> Software reliability is the probability that the software will execute for a particular period of time without a failure, weighted by the cost to the user of each failure encountered.

This definition is not necessarily based on the actual number of errors residing within the software system but is based on the impact that the errors have on the users. For example, a single error in a space shuttle control software is much more important than several errors in a matrix inversion software system which cause only "trivial" failures. Certainly, a software system which does not contain a serious error but has many trivial errors would generally be considered much more reliable than a system which does not contain the trivial errors but contains the single serious error.

The reliability of a software system is generally expected to grow as it evolves from the design stage to the coding stage and testing stages and down to the operational and maintenance stage. Modern software engineering practice advocates that testing should be performed as early as the design stage. Software errors detected in the design stage are easier and less expensive to remove than those detected during the testing or operational

stage. We also know that modern software design methodologies help in the likelihood of not committing errors in the design stage. Redundant programming, that is, implementing a software in different ways, is sometimes used to enhance software reliability. Fault tolerance programming is another popular technique. However, testing still remains the most commonly used approach to enhancing software reliability.

Testing for the presence of errors is usually done in stages [CHA78]:

1. First stage is the testing done at the module level by the implementing programmer.
2. Modules are then integrated forming a subsystem or the whole system is tested. The system is then tested. This is also known as alpha testing.
3. The software is then given to several "friendly users" who are willing to use the software in an operational environment and the problems encountered with the software are reported. This is known as beta testing.
4. Finally, software is released to all users and corrections are issued against it as problems are reported by the users.

This overall testing process coupled with the design testing process would, hopefully, result in an enhanced

reliability of the software system. Can the reliability of the software decrease as a result of the software correction (debugging) process? The answer is yes. This occurs when additional errors are accidentally injected into the system while removing some other errors.

Hopefully, with the use of better design methodologies, better documentation techniques, better programming languages, better testing strategies and better software management techniques, the likelihood of software reliability decreasing as the system evolves from the design to the operational stage will become less.

1.5 SOFTWARE RELIABILITY MODELS

Many studies have been undertaken during the last decade to analyze and study software failure data with the objective of finding ways that will lead to improved software performance. Such studies can be classified into one (or both) of two categories. In the first category, the emphasis is on the analysis of software failure data collected from small or large projects during development and/or operational phases. Studies in the second category are primarily aimed at the development of analytical models which are then used to obtain the reliability and other quantitative measures of software performance.

The analytical modelling work can then be classified into the following three major categories. The first one emphasizes the stochastic nature of software failures, while the second and the third use combinatorial analysis to provide measures of software reliability.

1. Failure Rate Based Models.
2. Combinatorial or Error-Seeding Models.
3. Input Domain Based Models.

1. Failure (Hazard) Rate Based Models: The times between indigenous errors or the number of indigenous errors observed during testing are used to estimate the shape of the hypothesized hazard function. From the estimated hazard function, one can estimate the number of errors remaining in the software, the mean-time-to-failure (MTTF) or the reliability of the software.

2. Combinatorial or Error Seeding Models: A known number of errors are seeded (planted) in the program. After testing, the number of exposed seeded and indigenous errors are counted. Using combinatorics and maximum likelihood estimation, estimates of the number of indigenous errors in the program or the reliability of the software can be estimated.

3. Input Domain Based Models: The basic approach here is to generate a set of test cases from an input (operational) distribution. Because of the difficulty in estimating the input distribution, the various models in this group partition the input domain into a set of equivalence classes. An equivalence class is usually associated with a program path. The reliability measure is calculated from the observed failures after execution (symbolic or physical) of the sampled test cases.

1.5.1 Failure Rate Based Models

Failure rate based models can be further classified as shown in Table 1.1.

The failure-rate (also known as hazard rate) function $z(t)$ is defined as the conditional probability that an error is exposed in the interval t to $t+\Delta t$, given that the error did not occur prior to time t [MYE76]. The reliability function $R(t)$ is the probability that no errors will occur from time zero to time t. Reliability theory tells us that $z(t)$ and $R(t)$ are related in the following form:

$$z(t) = [-dR(t)/dt]/R(t)$$

or

$$R(t) = \exp(-\int_0^t z(x)dx)$$

Also, mean-time-to-failute (MTTF) = $1/z(t)$.

Estimation of reliability, once the failure rate function $z(t)$ is known is thus straightforward. The failure rate based models given in Table 1.1 basically differ in their assumption on the failure rate function $z(t)$. Table 1.2 below displays these differences;

TABLE 1.1 TABLE OF FAILURE-RATED BASED SOFTWARE RELIABILITY MODELS

	Classical	Bayesian
Error Count Based Failure Rate Models	De-Eutrophication Process Model of Jelinski-Moranda [JEL72]	Littlewood Model [LIT80]
	Linear Function Testing Time Model of Schick and Wolverton [SCH78]	Goel-Okumoto Imperfect Debugging Model [GOE79]
	Parabolic Function Testing Time Model of Schick and Wolverton [SCH78]	
	Shooman Model [SHO72]	
	Shooman-Natarajan Model [DUN82]	
	Execution Time Model of Musa [MUS75]	
Non-Error Count Based Failure Rate Models	Non-Homogeneous Poisson Process Model of Goel & Okumoto [GOE79]	
	Geometric De-Eutrophication Process Model of Moranda [MOR75]	Littlewood and Verrall Model [LIT73]
	Geometric Poisson Process Model of Moranda [MOR75]	Thompson & Chelson Model [DUN82]
	Wagoner Model [DUN82]	

TABLE 1.2 SUMMARY OF FAILURE-RATE BASED MODELS

Model	Assumption on z(t)
De-Eutrophication Process Model	The software failure occurrence rate at any time t is assumed proportional to the number of errors remaining in the software, i.e., for the time interval between (i-1)st and ith failure, we have $Z(X_i) = \phi[N - (i-1)]$. N is the initial error content.
Schick-Wolverton Linear Failure Rate Model	Failure rate is assumed proportional to number of remaining errors in software and test time. For ith interval, $Z(X_i) = \phi[N - (i-1)]X_i$.
Schick-Wolverton Parabolic Failure-Rate Model	Failure rate is assumed proportional to residual errors and a parabolic function of test time. For ith interval, $Z(X_i) = \phi[N - (i-1)](-ax_i^2 + bx_i + c)$ $a,b,c > 0$
Shooman Model	$Z(t) = K[E_T/I_T - \int_0^\tau \rho(x)dx]$ where: K : proportionality constant E_T: total # errors I_T: total # instructions (object code) τ : debugging time $\rho(x)$: number of errors/total # instruction s/x debugging time. $\int_0^\tau \rho(x)dx$: total # of errors per I_T removed during time units of debugging time.
Shooman-Natarajan Model	$Z(t) = K\varepsilon_r(t)$ where: ε_r: number of remaining software faults. K : constant of proportionality

Execution Time Model of Musa

I. $z(\tau) = KfN_0 - Kfn(\tau)$ where:

- K : error exposure ratio
- f : linear execution frequency of program
- N_0: initial error content
- τ : CPU time utilized in operating the program
- $n(\tau)$: net number of errors corrected during τ.

II If $dn(\tau)/dt$ = error exposure rate, then $Z(\tau) = KfN_0 \exp[-Kf\tau]$

Non-Homogeneous Poisson Process (NHPP) Model of Goel & Okumoto

Assumes that the error detection rate $\lambda(t)$ is time dependent and is given by: $\lambda(t) = ab\exp[-bt]$ where:

- a: expected number of errors to be eventually detected
- b. error detection rate per error

Geometric De-Eutrophication Process Model of Moranda

Assume that the steps representing the decrease in failure rate between adjacent failure time are geometrically varying.

$Z(X_i) = DK^{i-1}$ where:

- D: initial error detection rate
- DK: error detection rate after the occurrence of the 1st error
- .
- .
- DK^{i-1}: error detection rate after the occurrence of the ith error.

Geometric Poisson Process Model of Moranda

A superposition of a geometric De-Eutrophication process and a Poisson process with parameter θ.

$$Z(X_i) = DK^{i-1} + \theta$$

Wagoner (Weibull) Model

$z(t) = \frac{\lambda}{\sigma}\left(\frac{t}{\sigma}\right)^{\lambda-1}$ where:

- σ : scale parameter
- λ : parameter to squeeze or stretch the shape of the distribution
- t : CPU time units

Goel-Okumoto Imperfect Debugging Model

The form of Z(t) is not obvious: however, the reliability function between the (k-1)st and Kth failure is;

$$R_K(x) = \sum_{j=0}^{K-1} \binom{K-1}{j} P^{K-j-1} Q^j \exp[-(N-K+j+1)]x$$

where:

P: Prob {of successful correction of a defect}

Q: Prob {of imperfect debugging}= 1-P

λ: Parameter of the exponential distributions governing the time between failures

N: Estimated initial number of defects.

Thompson & Chelson Model

$z(t) = \lambda$ but λ is treated as a random variable with a gamma density function

$$\frac{T_0(\lambda T_0)^{K_0}}{\Gamma(K_0 + 1)} \exp[- T_0]$$

where:

K_0: observed failures

T_0: testing time for K_0

K_0 and T_0 essentially represent previous testing experience.

Littlewood and Verrall Model

$z(t) = \lambda$ but λ is treated as a random variable distributed as Gamma with shape parameter α and scale parameter $\Psi(i)$, an increasing function of i.

Littlewood Model

$Z(X_i) = \lambda_i$ and λ_i is distributed as gamma $((N-i+1)\alpha, \beta + \sum_{1}^{i-1} t_j)$, where:

N-i+1: number of errors remaining when (i-1) failures have occurred.

t_j: execution time from (j-1)st failure to the jth failure.

α, β: parameters of gamma distribution.

As noted above a number of assumptions are made in the development of failure-rate models. A discussion of these assumptions is provided in the following paragraphs to point out the dangers in the use of these models when the assumptions are not satisfied. It should, however, be noted that some models could be robust to departures from many of these assumptions and can be used for reliability assessment purposes.

1. All the models described above assume that any error detected is immediately corrected. The correction process does not alter the program. All corrections remove the detected error (except the IDM model) and do not result in the introduction of new errors. It is not hard to accept that correction of a detected error in a program may result in new errors in the program. Goel and Okumoto [GOE79] tried to address the second limitation above by formulating the Imperfect Debugging Method (IDM). IDM assumes that the number of errors in the system at time t is governed by a Markov process. Time between transitions is exponentially distributed with rates dependent on the current error content of the program. The state transitions are governed by the probability of imperfect debugging. No one has yet addressed the problem in which the debugging process introduces new errors into the software.

2. Models such as those by Jelinski and Moranda, Musa, and Shooman assume that the software failure rate is a constant multiple of the number of remaining errors. This is the same as saying that each error in a given time interval (between failures) has the same chance of being detected. This, obviously, is not always true since errors that happen to reside in a portion of the code that is frequently executed by the user (or tested by the user) have a higher probability of being detected. Errors which reside in the unreachable (or never used) portion of the code will obviously have a lower (or zero) probability of being detected. Moranda tried to address this problem by reformulating the De-Eutrophication model into the Geometric De-Eutrophication Model and later to the Geometric Poisson Model. In these variations, the failure rate between adjacent failure intervals is geometrically varying.

The NHPP model by Goel and Okumoto also tried to alleviate these problems (i.e. problems with the constant failure rate models) by postulating a time dependent error detection rate model. Littlewood is more ambitious in trying to address this problem. He postulated a model which assumes that each remaining error in the program has a different rate of occurrence. The failure rate of the overall program is then just the sum of the individual error's rate of occurrence.

3. The Schick-Wolverton models happen to model a process where there is an increasing failure rate between failures. This may be a ridiculous assumption if we argue that software does not wear out. But there can be cases where the software failure rate might in fact increase and this may be attributed to the increased intensity of testing. This phenomenon is usually observed during the early stages of the software development cycle.

4. Basing the time between failures in terms of execution (CPU) time, as was assumed by Musa, Littlewood and Wagoner, may sometimes be unrealistic. An increase in accumulated time between two adjacent failures may not necessarily mean that the software has less and less number of errors or, putting it equivalently, that the software's reliability is improving. A very simple example will illustrate this point. Consider a program containing only a single error. The same copy of the program is given to two debuggers. One debugger spends a lot of time running and re-running the program (which can be very tempting to do on on-line and timesharing systems) trying to uncover the error. The second debugger, on the other hand, spends a lot of time analyzing the program before even attempting to make a test run. Suppose both debuggers are successful in finding the error. What is the resulting reliability of the software?

Execution time theory says that since the CPU time between failures of the first software is larger than that of the second software, then the first software is more reliable. Of course we know that this is not true since both software have the same reliability. Another example where execution time may be misleading is when a selected subset of the program is executed repeatedly. While the execution time is accumulating the test coverage is not and this will lead to an incorrect assessment of reliability. What is the most appropriate time unit to use for interfailure times is still a controversal topic.

5. What about the assumption of independence of interfailure times? Is this a realistic assumption? Chances are it is not. The testing process that is used to uncover errors is usually not a random process. The time to the next failure may very well depend on the nature and time to failure of the previous error. If the previous error was a very critical one, then we might decide to intensify the testing process and look for more critical errors. This intensification in the testing process may mean a shorter time to the next failure than what might have happened if the testing intensity were maintained at normal levels.

6. Most of the models require time between failure data to estimate reliability. There can be cases when the mean

time between failure is infinite; as such, these models become useless. The mean time between failures can be infinite if the user of the software has requirements that would only traverse the error free paths of the program.

7. Basing the reliability of the software on the remaining number of errors can also sometimes be ridiculous. A user does not really care whether a software has a certain number of remaining errors. As long as all his requirements are met correctly by the software, then as far as the user is concerned, the software is 100 percent reliable. Littlewood [LIT80] argued that a program with two bugs in little exercised portions of the code can be more reliable than a program with only one but frequently encountered bug.
8. All the models implicitly assume that the testing process, which generated the estimate for the failure rate, will be the same as the operating environment. This again is not true. A reliability measure conditioned on the user requirements rather than a simple unconditioned software reliability measure would be more realistic.
9. Some models assume that software reliability is time-dependent. Most software fail not because of the length of time it has been in use but fail because of the nature of the input to which it is subjected. Some software like real time control software or

operating systems show an illusion of failing with time-of-use because they are used almost continuously. In those environments, the time-dependence assumption may be valid.

10. Perhaps the most fundamental assumption is the treatment of the software as a black box. At least some software reliability models should take into consideration software characteristics and the characteristics of the software development process in addition to the failure times and the number of remaining errors.

1.5.2 Combinatorial or Error-Seeding Models

A number of combinatorial models have been proposed but the most popular (and most basic) is Mill's Hypergeometric Model. This model requires that a number of known errors be randomly inserted (seeded) in the program to be tested. The program is then tested for some amount of time. The number of original indigenous errors can be estimated from the numbers of indigenous and seeded errors uncovered during the test by using the hypergeometric distribution.

Let

n_0 = number of seeded errors

K = number of seeded errors detected during testing

N = total number of indigenous errors

r = number of indigenous errors detected during testing

We then have

$$P[\text{K seeded errors in r detected indigenous errors}] = \frac{\binom{n}{k}\binom{N-n}{r-k}}{\binom{N}{r}}$$

$$\text{MLE for } N = \frac{nr}{k}$$

A variant of the above model is the so-called Binomial model. Let q_i = Prob [errors] on each run i, then we have

$$\text{Prob[x errors in y trials]} = \binom{y}{x} q_i^x (1-q_i)^y.$$

The serious assumption of the above models is that the indigenous and seeded errors are assumed to have the same probability of being detected. In other words, the seeded errors must be of the same type and should have

the same distribution as the indigenous errors. This, of course, is difficult to meet in real-world conditions.

A suggestion is given in [HO78] to overcome this problem. In this improved approach, two teams are going to test the program independently. Suppose team 1 detects n errors and team 2 detects r errors, and the number of errors common to both teams is K. We can then view the errors detected by one team, say team 1, as seeded errors, and estimate the total number of indigenous errors N to be nr/K. Note, however, that simple errors may be discovered first and the distribution of errors detected may not resemble the actual distribution of errors; so that the estimates may be biased.

The advantage that is obvious with these combinatorial models over the failure-rate based models is that they are based on less and much simpler assumptions.

1.5.3 Input Domain Based Models

A good representative set of models in this group includes the Nelson (TRW) model [BRO75], Ho Model [HO78], and the Bastani model [BAS80].

Nelson (TRW) Model

The reliability of the software is measured by exposing (running) the software with a sample of n inputs. The n inputs are randomly chosen from the input domain set $E = (E_i: i = 1,N)$ where each E_i is the set of data values needed to make a run. The random sampling of n inputs is done according to a probability distribution P_i; the set $(P_i: i = 1,N)$ is the "operational profile" or simply user

input distribution. If n_e is the number of inputs that resulted in execution failure, then an unbiased estimate for the software reliability $\hat{R}$ is $1 - (n_e/n)$. However, it may be the case that the test set used during the verification phase may not be representative of the expected operational usage. Brown and Lipow [BRO75] suggested an alternative formula for $\hat{R}$ which is

$$\hat{R} = 1 - \sum_{i=1}^{N} \frac{f_j}{n_j} P(E_j)$$

where

n_j = number of runs sampled from input subdomain E_j

f_j = number of failures observed out of n_j runs.

The main difference between Nelson's $\hat{R}$ and Brown and Lipow's $\hat{R}$ is that the former explicitly incorporates the usage distribution or the test case distribution while the latter implicitly assumes that the accomplished testing is representative of the expected usage distribution. Both models assume prior knowledge of the operational usage distribution. This may not be easy to do for some real-world software. Another criticism of this approach is the use of random testing.

Ho Model

Reliability estimation in this model proceeds by first generating the symbolic execution tree of the program. This tree characterizes all the execution paths and their associated outputs in the program. The nodes represent statements while the edges represent the state vector resulting from symbolic execution along the path

from the root statement to the current statement. A procedure for generating the symbolic execution tree is given in [H078]:

I. The first statement is the root of the tree.

II. If a leaf is not a STOP or RETURN statement, symbolically execute the statement corresponding to the node. If the current statement is a conditional statement, the feasibility of the branches is examined. New nodes are created for statements which are successors of the current statement. Edges, labelled with state vectors are joined between the current node and the new node(s).

III. Go to II.

The generated execution paths from the symbolic execution tree are proven correct or are sample tested. For a given path, say path i, if it is proven correct, then the path reliability $R_i = 1$. If path i cannot be proven correct, a random sample of N test cases is generated that will execute path i. If no failures result from the execution of the N test cases, then R_i is bounded below by $1 - C_i$ where C_i is the confidence interval of path i. The length of C_i is a function of our given confidence coefficient α. On the other hand, if n

failures are observed and the errors not corrected, then R_i is bounded below by $\frac{N - n}{N} - C_i$. If the observed n failures are corrected, then the sample testing is repeated for path i.

Finally, the software reliability estimate R is calculated from

$$R = \sum_{i=1}^{m} f_i R_i$$

where:

f_i = weighting factor or path i which corresponds to the execution frequency of path i.

m = total number of execution paths.

One difficulty with applying this approach is the large number of paths that may exist for real world software.

Bastani Model

This input domain based model estimates the reliability R from the relation

$$\hat{R} = 1 - \hat{V}_{e_r}$$

where:

$\hat{V}_{e_r}$ = the total error size remaining in the program.

$\hat{V}_{e_r}$ can be determined by testing the program and locating and estimating the size of errors found [BAS80]. An error

has a large size if it is easily detected (i.e., if it affects many input elements). An error has a small size if it is relatively difficult to detect. The size of an error depends on the way test inputs are selected. Good test case selection strategies like path testing, boundary value analysis, magnify the size of an error since they exercise error-prone constructs. The observed error size is lower if random testing is employed. Although the model does not assume random testing (in fact, any test strategy can be employed), it offers no easy or systematic way to estimate V_{e_r}.

SECTION 2

A TIME DEPENDENT FAULT DETECTION RATE MODEL

2.1 INTRODUCTION

In this section, our objective is to develop a parsimonious model whose parameters have a physical interpretation, and which can be used to predict various quantitative measures for software performance assessment. Also of interest is the applicability of the model over a broad class of projects. Further, it should be possible to estimate the parameters of the model from available failure data which could be given as either the number of failures in specified time intervals, or as times between software failures.

With this objective, we develop and investigate a nonhomogeneous Poisson process (NHPP) [BRO72] model with a time dependent fault detection rate for the software failure phenomenon. By studying the behavior of the cumulative number of failures by time t process, N(t) it is shown in section 2.2 that this process can be well described by a non-homogeneous Poisson process (NHPP) with a two parameter exponentially decaying fault detection rate.

NHPP has been used by many researchers to describe random phenomena in various applications [CRO74, DUN75, DVA64]. Some such applications are the occurrences of coal mining disasters [MAG52]; equipment failures [DUA64, LEW64, PRO63]; transactions in a data-base system [LEW76]; and software error counts over a series of time intervals [SCH75]. Various forms of the intensity function for the NHPP used in actual applications are the exponential polynomial rate function [LEW76], a log-linear rate function [COX66], and a Weibull rate function [CRO74, DON75, MOE76].

Several measures for software performance assessment, such as the number of faults remaining in the system, distribution of time to next failure, and software reliability, are proposed in section 2.3. Based on the NHPP model, expressions are then derived for obtaining the estimates and confidence limits for these performance measures.

Two methods are described in section 2.4 for estimating the parameters of the model from available failure data. The first one is for the case when data is given in the form of number of failures in given time intervals. The time intervals can be of equal or un-

equal lengths, but the data must be converted to an interval of the same length. The second method is used when times between software failures are given. Analyses of actual failure data are presented in section 2.5.

2.2 MODEL DEVELOPMENT

A software system in use is subject to failures caused by faults present in the system. The faults are encountered when a sequence of instructions is executed which, in turn, depends on the input data set. In this section, we develop a model to describe this failure occurrence phenomenon.

2.2.1 Deterministic Analysis of Software Failure Process

It is useful to first make a simpler analysis by ignoring the statistical fluctuations in the number of software failures before analyzing the failure phenomenon as a stochastic process [COX65]. Let $n(t)$ denote the cumulative number of software failures detected by time t. Assume that $n(t)$ is large enough so that is can be expressed as a continuous function of t. Since the number of errors in a system is a finite value, $n(t)$ is a bounded non-decreasing function of t with

$$n(0) = 0$$

and (2.1)

$$n(\infty) = a$$

For purposes of modeling, we assume that the usage of the system is basically similar over time. Then the number of failures in $(t, t+\Delta t)$ is proportional to the number of undetected faults at t, i.e.,

$$n(t+\Delta t) - n(t) = b\{a-n(t)\}\Delta t \ , \qquad (2.2)$$

where b is a proportionality constant.

A graphical representation of the above description is provided in Figure 2.1.

Now, from Equation (2.2), we get the differential equation

$$n'(t) = ab - bn(t) \ . \qquad (2.3)$$

Taking the Laplace transform [ABR65, BUC56] of Equation (2.3) under the conditions of Equation (2.1), we have

$$s\tilde{n}(s) = \frac{ab}{s} - b\tilde{n}(s) \ ,$$

or

$$\tilde{n}(s) = \frac{ab}{s(s+b)} \ , \qquad (2.4)$$

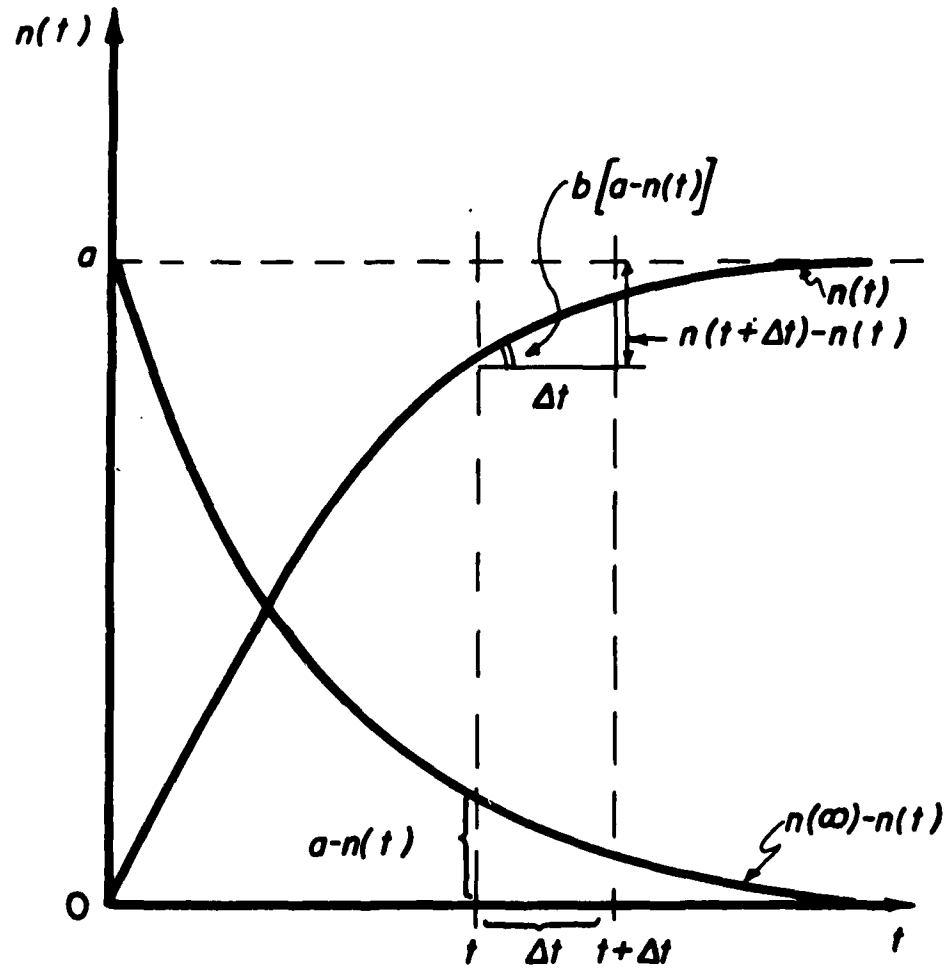


FIGURE 2.1. A Graphical Representation of the Deterministic Model for Software Failures.

where

$$\tilde{n}(s) = \int_0^{\infty} e^{-st} \cdot dn(t) \ . \qquad (2.5)$$

The solution of Equation (2.3) is thus obtained by inverting Equation (2.4) and is given by

$$n(t) = a(1 - e^{-bt}) \ . \qquad (2.6)$$

Under the assumptions discussed above, Equation (2.6) is the deterministic model of the software failure process. For given a and b, we can easily compute the number of failures to be encountered by some time t so that the failure phenomenon can be described with certainty. It should be noted, however, that the actual failure phenomenon is not deterministic.

2.2.2 Stochastic Analysis of Software Failure Process

In an actual usage, the software system is subjected to random inputs causing the failures to occur at random times, i.e., the failure phenomenon is stochastic (non-deterministic). Therefore, a realistic description of the failure process must incorporate this randomness.

Let $(N(t), t \geq 0)$ be a counting process [PYK61, ROS76, SNY75] representing the cumulative number of failures by time t. (Note that N(t) is a random variable while n(t) above was taken to be deterministic.) Assuming that each failure is caused by one fault, N(t) also represents the cumulative number of faults detected by time t. It should be pointed out that a detected fault may not be removed and, as a result, may cause additional failure(s) at a later stage. For the N(t) process, such recurrences are counted as new events.

Let m(t) be the mean value function of the N(t) process, i.e.,

$$m(t) \equiv E[N(t)] \ . \tag{2.7}$$

Since m(t) represents the expected number of software failures or detected faults by time t, it is a non-decreasing function of t. If we assume that there will be a finite number of faults to be detected over a long period of time, m(t) has the following boundary conditions:

$$m(t) = \begin{cases} 0, & t = 0 \\ \\ a, & t = \infty \end{cases} \tag{2.8}$$

where $a < \infty$ and represents the expected number of software faults to be eventually detected. Furthermore, it is assumed that, for small Δt, the expected number of software failures during $(t, t+\Delta t)$ is proportional to the expected number of software faults undetected by time t, i.e.,

$$m(t+\Delta t) - m(t) = b\{a-m(t)\}\Delta t \ , \qquad (2.9)$$

where b is a constant of proportionality. Solving the differential equation obtained from Equation (2.9) under the boundary conditions of Equation (2.8), we get

$$m(t) = a(1 - e^{-bt}) \ . \qquad (2.10)$$

This equation specifies the mean value function for the underlying software failure counting process N(t). The intensity function, obtained by taking the derivative of m(t), represents the fault detection rate at time t and is given by

$$\lambda(t) \equiv m'(t) = abe^{-bt} \ . \qquad (2.11)$$

We now study the probabilistic behavior of the N(t) process by using m(t) and $\lambda(t)$. Since there are no failures at t = 0, we have N(0) = 0. It is also reasonable to assume that the number of software failures during non-overlapping time intervals are independent. In other words, for any finite collection of times $t_1 < t_2 < \ldots < t_n$, the n random variables $N(t_1)$, $\{N(t_2)-N(t_1)\},\ldots,\{N(t_n)-N(t_{n-1})\}$ are statistically independent. This implies that the counting process $\{N(t),\ t \geq 0\}$ has independent increments.

We assign the probabilities on the increments of the N(t) process as follows.

$$N(t+\Delta t)-N(t) = \begin{cases} 0 \text{ with probability } 1-\lambda(t)\Delta t+O(\Delta t) \\ 1 \text{ with probability } \lambda(t)\Delta t+O(\Delta t) \\ 2 \text{ with probability } O(\Delta t) \\ \vdots \qquad\qquad \vdots \end{cases} \tag{2.12}$$

where

$$\frac{O(\Delta t)}{\Delta t} \to 0 \quad \text{as} \quad \Delta t \to 0 \ .$$

The underlying N(t) process satisfying conditions of Equation (2.12) is now a NHPP with mean value function m(t) and intensity function $\lambda(t)$ as given in Equations (2.10) and (2.11), respectively [FELW57, FELW60]. Hence, the distribution of N(t) is given by

$$P\{N(t) = y\} = \frac{\{m(t)\}^y}{y!} e^{-m(t)}, \quad y = 0,1,2,... \quad (2.13)$$

Under the assumptions discussed above, the stochastic behavior of the software failure phenomenon can be completely described by Equation (2.13). It should be pointed out that Equation (2.9) implies that the ratio

$$\frac{\text{Number of faults detected during } (t, t+\Delta t)}{(\text{Number of faults undetected by } t)\Delta t} = b \quad (2.14)$$

is constant at any time t. Therefore, b can be interpreted as the error detection rate per error.

Equations (2.10) and (2.13) constitute the basic software failure model under study in this report.

2.3 SOFTWARE PERFORMANCE MEASURES

The model developed in section 2.2 is a description of the failure phenomenon. In order to use this model to predict software performance, we generally need expressions for quantitative measures, such as the number of failures by some prespecified time, the number of faults remaining in the software at a future time, and software reliability during a mission. In this section, we develop models that can be employed to estimate such quantities.

2.3.1 Number of Software Faults Detected by t

For given a and b, the distribution of N(t), the cumulative number of software failures detected by time t, is obtained from Equations (2.10) and (2.13) as

$$P\{N(t)=y\} = \frac{\{a(1-e^{-bt})\}^y}{y!} e^{-a(1-e^{-bt})}, \quad y = 0,1,2,\ldots . \qquad (2.15)$$

In other words, N(t) has a Poisson distribution with mean

$$m(t) \equiv E[N(t)] = a(1 - e^{-bt}) . \qquad (2.16)$$

Note that

$$P\{N(\infty)=y\} = \frac{a^y}{y!} e^{-a} \quad , \; y = 0,1,2,\ldots, \qquad (2.17)$$

i.e., the distribution of $N(\infty)$, the total number of failures encountered or faults detected if the system is used indefinitely, is also a Poisson distribution with mean 'a'. This result is consistent with theoretical studies which indicate that the Poisson process is the limiting distribution of many phenomena similar to the software error occurrence phenomenon [MIL76, SNY 75].

2.3.2 Number of Remaining Faults

We have been considering the number of failures encountered by time t, $N(t)$. Since many of the performance measures depend on the number of faults remaining in the system, we now consider this phenomenon.

Let $\overline{N}(t)$ be the number of faults remaining in the system at time t, i.e.,

$$\overline{N}(t) \equiv N(\infty) - N(t) \; . \qquad (2.18)$$

The expectation of $\bar{N}(t)$ is given by

$$E[\bar{N}(t)] = ae^{-bt} . \qquad (2.19)$$

2.3.3 Conditional Distribution and Expectation of $\bar{N}(t)$

If we have already observed y faults, it is useful to know the distribution of the number of faults yet to be detected. In other words, the conditional distribution of $\bar{N}(t)$, given that $N(t) = y$, is

$$P\{\bar{N}(t)=x \mid N(t)=y\} = \frac{P\{\bar{N}(t)=x,\ N(t)=y\}}{P\{N(t)=y\}} . \qquad (2.20)$$

Now the event $\bar{N}(t) = x$ denotes occurrences over the time interval (t,∞) while the event $N(t) = y$ denotes occurrences over the interval $(0,t)$, i.e., these two events represent non-overlapping time intervals. From a basic property of the NHPP process, such events are independent of each other, so that we have

$$P\{\bar{N}(t)=x \mid N(t)=y\} = P\{\bar{N}(t)=x\}, \quad x = 0,1,2,... \qquad (2.21)$$

or

$$P\{N(\infty)-N(t)=x|N(t)=y\} = \frac{\{m(\infty)-m(t)\}^x}{x!} e^{-\{m(\infty)-m(t)\}}.$$

Or, substituting for $m(\infty)$ and $m(t)$ from Equation (2.10), we get

$$P\{N(\infty)-N(t)=x|N(t)=y\} = \frac{\{a-a(1-e^{-bt})\}^x}{x!} e^{-\{a-a(1-e^{-bt})\}}.$$

This yields

$$P\{\overline{N}(t)=x|N(t)=y\} = \frac{\{ae^{-bt}\}^x}{x!} e^{-ae^{-bt}} . \quad (2.22)$$

Finally, the expected number of faults to be detected, given $N(t) = y$, is

$$E[\overline{N}(t)|N(t)=y] = ae^{-bt} . \quad (2.23)$$

This conditional distribution is important for deciding whether the software system under development can be released or not. The decision should be made based on the number of faults remaining in the software because this quantity plays an important role in software reliability assessment. Suppose that the decision-maker conducts an experiment and finds y software faults by time t. Then, a decision might be to

$$\text{Accept if } \bar{N}(t) \leq n_0$$

and

$$\text{Reject if } \bar{N}(t) > n_0 \ ,$$

where n_0 is some specified number. For this decision rule, the probability that the software system is accepted for a given number of failures y by time t is

$$P\{\text{Accept}\} = P\{\bar{N}(t) \leq n_0 \,|\, N(t) = y\}$$

and, using Equation (2.22), becomes

$$P\{\text{Accept}\} = \sum_{i=0}^{n_0} P[\bar{N}(t)=i \,|\, N(t)=y] \ . \qquad (2.24)$$

The conditional expectation of $\bar{N}(t)$, given $N(t)=y$, is given by

$$E[\bar{N}(t) \,|\, N(t)=y] = E[\bar{N}(t)]$$

or

$$E[\bar{N}(t) \,|\, N(t)=y] = ae^{-bt} \ . \qquad (2.25)$$

Therefore, the expected number of faults remaining in the software system at time t, given that y errors have been detected during the testing period t, is simply the expected number of faults to be detected during $[t,\infty]$.

2.4 SOFTWARE RELIABILITY AND DISTRIBUTION OF TIME BETWEEN FAILURES

2.4.1 Software Reliability

Let a sequence of random variables $\{X_i, i = 1,2,...\}$ denote a sequence of times between software failures associated with the N(t) process. Then X_i denotes the time between the (i-1)st and the ith failures. We also define

$$S_n \equiv \sum_{i=1}^{n} X_i, \quad n = 1,2,... \tag{2.26}$$

which represents the time to the nth failure. Let $\Phi_1(x)$ be the Cumulative Distribution Function (cdf) of X_1, i.e.,

$$\Phi_{X_1}(x) \equiv P\{X_1 \leq x\} . \tag{2.27}$$

Note that the event $\{X_1 > x\}$ implies that there are no failures during (0,x], i.e., the event $\{N(x) = 0\}$. Then, using Equation (2.15), the reliability function associated with the first failure time is given by

$$R_{X_1}(x) \equiv P\{X_1 > x\} = P\{N(x) = 0\}$$

or

$$R_{X_1}(x) = e^{-a(1 - e^{-bx})} \quad . \qquad (2.28)$$

Now, the cdf of X_1 can be written as

$$\Phi_{X_1}(x) = 1 - R_{X_1}(x)$$

or

$$\Phi_{X_1}(x) = 1 - e^{-a(1 - e^{-bx})} \quad . \qquad (2.29)$$

The Probability Density Function (pdf) is defined as

$$\phi_{X_1}(x) = \frac{d}{dx} \Phi_{X_1}(x)$$

so that

$$\phi_{X_1}(x) = abe^{-bx}e^{-a(1 - e^{-bx})} \quad . \qquad (2.30)$$

Next, consider the conditional probability distribution, $\Phi_{X_2|X_1}(x|s)$, of $\{X_2|X_1\}$. The event $\{X_2 > x|X_1 = s\}$ implies {no failures in (s,s+x]}. Then the conditional

reliability function of the second failure, given that the first failure occurs at time s, is given by

$$R_{X_2|X_1}(x|s) \equiv P\{X_2 > x|X_1 = s\}$$

$$= P\{\text{no failures in } (s,s+x]\}$$

$$= P\{N(s+x) - N(x) = 0\}$$

$$= e^{-[m(s+x) - m(s)]}$$

$$= e^{-a[e^{-bs} - e^{-b(s+x)}]} . \qquad (2.31)$$

From Equation (2.31), we obtain

$$\Phi_{X_2|X_1}(x|s) \equiv 1 - R_{X_2|X_1}(x|s)$$

$$= 1 - e^{-a[e^{-bs} - e^{-b(s+x)}]} \qquad (2.32)$$

and

$$\phi_{X_2|X_1}(x|s) \equiv \frac{d}{dx} \Phi_{X_2|X_1}(x|s)$$

$$= abe^{-b(s+x)}e^{-a\{e^{-bs}-e^{-b(s+x)}\}}. \qquad (2.33)$$

Combining Equations (2.29) and (2.33), we get the joint density of X_1 and X_2 as

$$\phi_{X_1,X_2}(x_1,x_2) = \phi_{X_2|X_1}(x_2|x_1)\phi_{X_1}(x_1)$$

$$= (abe^{-bx_1})(abe^{-b(x_1+x_2)})$$

$$\cdot e^{-a(1-e^{-bx_1})}e^{-a\{e^{-bx_1}-e^{-b(x_1+x_2)}\}}$$

or

$$\phi_{X_1,X_2}(x_1,x_2) = a^2b^2e^{-bx_1}e^{-b(x_1+x_2)}e^{-a\{1-e^{-b(x_1+x_2)}\}}. \quad (2.34)$$

Making the transformation $s_1 = x_1$, $s_2 = x_1+x_2$, the joint density of S_1 and S_2 is

$$\phi_{S_1,S_2}(s_1,s_2) = a^2b^2e^{-bs_1}e^{-bs_2}e^{-a(1-e^{-bs_2})}. \quad (2.35)$$

In general, it can be shown that the conditional reliability function of X_k, given $S_{k-1} = s$, is given by

$$R_{X_k|S_{k-1}}(x|s) = e^{-a\{e^{-bs} - e^{-b(s+x)}\}}. \quad (2.36)$$

2.4.2 Conditional Distribution of $X_k|S_{k-1}$

The conditional cdf and pdf are obtained from Equation (2.36) by recalling that $R(x) = 1 - \Phi(x)$ and $\phi(x) = \frac{d}{dx}\Phi(x)$. Thus, we have

$$\Phi_{X_k|S_{k-1}}(x|s) = 1 - e^{-a\{e^{-bs} - e^{-b(s+x)}\}} \qquad (2.37)$$

and

$$\phi_{X_k|S_{k-1}}(x|s) = abe^{-b(s+x)}e^{-a\{e^{-bs}-e^{-b(s+x)}\}}, \qquad (2.38)$$

respectively.

As can be seen from the above equations, the time to the next failure depends on the time when the last failure occurs. It should be noted that the distributions of times between failures are improper, i.e.,

$$\Phi_{X_k|S_{k-1}}(\infty|\,.) = 1 - e^{-ae^{-bs}} < 1 \; . \qquad (2.39)$$

This is due to the fact that the event {no failures in $(0,\infty]$} is allowed in our model. Hence, the expectations of these quantities do not exist.

2.4.3 Joint Density of Waiting Times

As defined above, $\{X_k,\ k = 1,2,\ldots\}$ denotes the sequence of times between software failures. Then

$$S_n = \sum_{i=1}^{n} X_i, \quad n = 1,2,\ldots$$

is called the waiting time to the nth software failure. This quantity is quite important for estimation of parameters a and b and, hence, we obtain the distribution of $\{S_1,S_2,\ldots,S_n\}$. The distribution is obtained by using an approach similar to that used for getting Equation (2.34). The result is summarized in the following theorem.

Theorem. The joint probability density of $S_1,S_2,\ldots,S_n$ is given by

$$\phi_{S_1,\ldots,S_n}(s_1,\ldots,s_n) = (ab)^n \cdot e^{-b\sum_{i=1}^{n} s_i} \cdot e^{-a(1-e^{-bs_n})} \quad (2.40)$$

where $s_1,s_2,\ldots,s_n$ denote the realizations of $S_1,S_2,\ldots,S_n$, respectively.

The density can also be written as

$$\phi_{S_1,\ldots,S_n}(s_1,\ldots,s_n) = e^{-m(s_n)} \prod_{k=1}^{n} \lambda(s_k) \quad (2.41)$$

where $\lambda(s_k) = \frac{d}{ds_k}\{m(s_k)\}$ and $m(s_k) = a(1-e^{-bs_k})$. For a proof of this theorem, see [COX66] and [DON75].

Equation (2.40) will be used later to estimate a and b based on observed data $\underline{s} = (s_1,\ldots,s_n)$.

2.4.4 Joint Counting Probability

The property of independent increments, along with Equations (2.8) and (2.12), provides a complete statistical characterization for NHPP so that the joint counting probability can be determined for any collection of times $0 < t_1 < t_2 < \ldots < t_n$. That is, with $t_0 = 0$, $y_0 = 0$.

$$P\{N(t_1) = y_1, N(t_2) = y_2, \ldots, N(t_n) = y_n\}$$

$$= \prod_{i=1}^{n} P\{N(t_i) - N(t_{i-1}) = y_i - y_{i-1}\}$$

$$= \prod_{i=1}^{n} \frac{[m(t_i) - m(t_{i-1})]^{y_i - y_{i-1}}}{(y_i - y_{i-1})!} e^{-m(t_n)}. \quad (2.42)$$

Equation (2.42) is needed for estimating the parameters a and b for given data $\{(y_i,t_i),\ i = 1,2,\ldots,n\}$.

2.5 ESTIMATION OF MODEL PARAMETERS FROM FAILURE DATA

The basic models for the failure process and performance measures were developed in Sections 2.2 and 2.3, respectively. In order to use these models for software performance assessment, the only parameters to be specified are the total expected number of errors to be detected, a, and the error detection rate per error, b. In other words, for given a and b, various useful quantities can be computed from the relevant equations in sections 2.2 and 2.3.

In general, a and b are not known for a specific software system and are estimated from the available data generated during testing. However, that is not the only way to get a and b. One may be willing to extrapolate these values based on the data from one or more "similar" systems. Another method would be to use a Bayesian approach, whereby knowledge about a and b can be expressed as prior distributions and used for performance assessment. This approach can also be used in conjunction with available data and is specially useful when failure data are scarce or expensive to collect.

The purpose of this section is to describe methods for estimating a and b from failure data. Use of these

methods is illustrated later via failure data from operational systems. Such data are generally available as

(i) total number of failures in given time intervals; and/or as

(ii) times between failures.

Most of the available data is given in the form of number of failures in given time intervals; the data on times between failures is very rare. Nevertheless, both of these cases are considered below.

2.5.1 Estimation When Cumulative Failures Are Given

We first consider the case when data are available as cumulative number of failures in given time intervals. Suppose $y_1, y_2, \ldots, y_n$ are the cumulative number of failures detected by times $t_1, t_2, \ldots, t_n$, respectively. This can also be written as data pairs $\{(y_i, t_i),\ i = 1,2,\ldots,n\}$. Thus, the number of failures in time interval (t_{i-1}, t_i) is $(y_i - y_{i-1})$ for $i = 1,2,3,\ldots,n$, where $t_0 = 0$ and $y_0 = 0$. We will obtain the Maximum Likelihood Estimates $\hat{a}$ and $\hat{b}$ of a and b, respectively. To do this, we first write the joint density and obtain the likelihood function, and then the log-likelihood function. Next, we take the partial derivatives of the log-likelihood function with respect to a and b and equate them to zero for maximization.

The solutions of the resulting two equations are the desired values $(\hat{a},\hat{b})$.

Now, to get the joint density, we note that in our notations $y_1,y_2,\ldots,y_n$ are the observed values of $N(t_1)$, $N(t_2),\ldots,N(t_n)$, respectively. Hence, from Equation (2.42),

$$P\{N(t_1) = y_1, N(t_2) = y_2, \ldots, N(t_n) = y_n\}$$

$$= \prod_{i=1}^{n} \frac{[m(t_i)-m(t_{i-1})]^{y_i-y_{i-1}}}{(y_i-y_{i-1})!} e^{-\{m(t_i)-m(t_{i-1})\}} \tag{2.43}$$

where $m(t_i) = a(1-e^{-bt_i})$.

It is well known that the likelihood function for the parameters is simply the joint density of $y_1,y_2,\ldots,y_n$, with these values considered as known constants. Substituting for $m(t_i)$ in Equation (2.43), the likelihood function for (a,b), given the data $(\underline{t},\underline{y})$, is

$$L(a,b|\underline{y},\underline{t}) = \prod_{i=1}^{n} \frac{\{a(e^{-bt_{i-1}}-e^{-bt_i})\}^{y_i-y_{i-1}}}{(y_i-y_{i-1})!}$$

$$\cdot\, e^{-a(1-e^{-bt_n})} . \tag{2.44}$$

Taking the natural logarithm of Equation (2.44) yields:

$$\ell nL(a,b|\underline{y},\underline{t}) = \sum_{i=1}^{n} (y_i - y_{i-1}) \ell na + \sum_{i=1}^{n} (y_i - y_{i-1}) \cdot \ell n(e^{-bt_{i-1}} - e^{-bt_i}) - \sum_{i=1}^{n} \cdot \ell_n (y_i - y_{i-1})! - a(1-e^{-bt_n}). \quad (2.45)$$

As mentioned above, the maximum likelihood estimates (mle's) are those values of a and b which maximize $\ell nL(a,b|\underline{t},\underline{y})$, i.e., which satisfy (for brevity we write L to denote $L(a,b|\underline{t},\underline{y})$)

$$\frac{\partial \ell nL}{\partial a} = 0$$

and (2.46)

$$\frac{\partial \ell nL}{\partial b} = 0 \ .$$

By taking the partial derivatives of Equation (2.45) and equating them to zero, we obtain, after some simplification (recall that $y_0 = 0$),

$$a(1-e^{-bt_n}) = y_n \ , \quad (2.47)$$

and

The solutions of the resulting two equations are the desired values $(\hat{a},\hat{b})$.

Now, to get the joint density, we note that in our notations $y_1,y_2,\ldots,y_n$ are the observed values of $N(t_1)$, $N(t_2),\ldots,N(t_n)$, respectively. Hence, from Equation (2.42),

$$P\{N(t_1) = y_1, N(t_2) = y_2, \ldots, N(t_n) = y_n\}$$

$$= \prod_{i=1}^{n} \frac{[m(t_i)-m(t_{i-1})]^{y_i-y_{i-1}}}{(y_i-y_{i-1})!} e^{-\{m(t_i)-m(t_{i-1})\}} \tag{2.43}$$

where $m(t_i) = a(1-e^{-bt_i})$.

It is well known that the likelihood function for the parameters is simply the joint density of $y_1,y_2,\ldots,y_n$, with these values considered as known constants. Substituting for $m(t_i)$ in Equation (2.43), the likelihood function for (a,b), given the data $(\underline{t},\underline{y})$, is

$$L(a,b|\underline{y},\underline{t}) = \prod_{i=1}^{n} \frac{\{a(e^{-bt_{i-1}}-e^{-bt_i})\}^{y_i-y_{i-1}}}{(y_i-y_{i-1})!}$$

$$\cdot e^{-a(1-e^{-bt_n})} . \tag{2.44}$$

Taking the natural logarithm of Equation (2.44) yields:

$$\ell nL(a,b|\underline{y},\underline{t}) = \sum_{i=1}^{n} (y_i - y_{i-1}) \ell na + \sum_{i=1}^{n} (y_i - y_{i-1}) \cdot \ell n(e^{-bt_{i-1}} - e^{-bt_i}) - \sum_{i=1}^{n} \cdot \ell_n (y_i - y_{i-1})! - a(1-e^{-bt_n}). \quad (2.45)$$

As mentioned above, the maximum likelihood estimates (mle's) are those values of a and b which maximize $\ell nL(a,b|\underline{t},\underline{y})$, i.e., which satisfy (for brevity we write L to denote $L(a,b|\underline{t},\underline{y})$)

$$\frac{\partial \ell nL}{\partial a} = 0$$

and (2.46)

$$\frac{\partial \ell nL}{\partial b} = 0 \ .$$

By taking the partial derivatives of Equation (2.45) and equating them to zero, we obtain, after some simplification (recall that $y_0 = 0$),

$$a(1-e^{-bt_n}) = y_n \ , \quad (2.47)$$

and

$$at_n e^{-bt_n} = \sum_{i=1}^{n} \frac{(y_i - y_{i-1})(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{e^{-bt_{i-1}} - e^{-bt_i}} . \quad (2.48)$$

As can be easily seen, all the quantities in Equations (2.47) and (2.48) are known except a and b, which are to be estimated. These equations do not yield simple analytical forms and we use numerical methods for their solution. The resulting values of a and b are the mle's $\hat{a}$ and $\hat{b}$, respectively.

It should be pointed out that, even though the mle's are the desired values, it is often useful to study the log-likelihood surface as a function of parameters a and b. For given data, a plot of the log-likelihood surface can be obtained by solving Equation (2.45) for a grid of values of a and b. If the plot is flat, it woull indicate a large variability associated with the mle's while a sharp surface is an indicator of low variability. A surface with sharp rises and falls might cause problems in numerical solution of Equations (2.47) and (2.48), while a well-behaved surface would ensure rapid convergence to the values $\hat{a}$, $\hat{b}$.

2.5.1.1 Confidence Region for (a,b)

In addition to the mle's $\hat{a}$, $\hat{b}$, we generally want to quantify the region in which the true values a, b might lie with a specified degree of confidence. This is referred to as obtaining the $100(1-\alpha)\%$ joint confidence region for (a,b). In general, it is not possible to get the exact confidence region [FIN76] because the true distribution of $(\hat{a},\hat{b})$ is unknown. However, mle's have a very desirable property that they are asymptotically normally distributed, if the sample size is large.

Also of great usefulness is the invariance property of the mle's, i.e., a function of (a,b) can be estimated by using the mle's $\hat{a}$, $\hat{b}$ and this function will also be a mle. This will be useful for estimating $\overline{N}(t)$, $R(t)$, etc.

Formally, as indicated above, the mle's are normally distributed for large n, i.e.,

$$\begin{pmatrix} \hat{a} \\ \hat{b} \end{pmatrix} \sim N \left(\begin{pmatrix} a \\ b \end{pmatrix} , \Sigma_{cov} \right) \quad \text{as } n \to \infty . \quad (2.49)$$

The variance-covariance matrix represents

$$\Sigma_{cov} = \begin{pmatrix} \text{Var}(a) & \text{Cov}(a,b) \\ \text{Cov}(b,a) & \text{Var}(b) \end{pmatrix}$$

and is given by

$$\Sigma_{cov} = \begin{pmatrix} r_{aa} & r_{ab} \\ r_{ba} & r_{bb} \end{pmatrix}^{-1} \qquad (2.50)$$

where

$$r_{ij} = -E\frac{\partial^2 \ell nL}{\partial i \partial j} , \quad i,j = a,b .$$

That is,

$$r_{aa} = -E \frac{\partial^2 \ell nL}{\partial a^2} \qquad (2.51)$$

$$r_{ab} = r_{ba} = -E \frac{\partial^2 \ell nL}{\partial a \partial b} \qquad (2.52)$$

$$r_{bb} = -E \frac{\partial^2 \ell nL}{\partial b^2} \qquad (2.53)$$

Taking the appropriate partial derivatives of Equation (2.45) and substituting in Equations (2.51), (2.52), and (2.53), we obtain, after some simplification, (recall that $E[N(t_i)] \equiv m(t_i) = a(1-e^{-bt_i})$):

$$r_{aa} = \frac{1}{a} \sum_{i=1}^{n} (e^{-bt_{i-1}} - e^{-bt_i}) , \qquad (2.54)$$

$$r_{ab} = r_{ba} = t_n e^{-bt_n} , \qquad (2.55)$$

and

$$r_{bb} = a \sum_{i=1}^{n} \frac{(t_{i-1}-t_i)^2 e^{-bt_{i-1}-bt_i}}{(e^{-bt_{i-1}} - e^{-bt_i})} - at_n^2 \cdot e^{-bt_n} . \qquad (2.56)$$

Substituting these expressions in Equation (2.50), we get the variance-covariance matrix for $(\hat{a},\hat{b})$. Thus, the asymptotic distribution of $(\hat{a},\hat{b})$ is completely specified if (a,b) are known. However, in practice, (a,b) are not known. Therefore, we use their estimates, $(\hat{a},\hat{b})$, in Equations (2.49), (2.54), (2.55), and (2.56) to get estimates of the parameters of the asymptotic bivariate normal distribution.

Now, the correlation coefficient between $\hat{a}$ and $\hat{b}$ is estimated as

$$\rho_{\hat{a},\hat{b}} = \frac{\text{Cov}(\hat{a},\hat{b})}{\sqrt{\text{Var}(\hat{a}),\ \text{Var}(\hat{b})}} \quad , \tag{2.57}$$

where $\text{Var}(\hat{a})$, $\text{Var}(\hat{b})$, $\text{Cov}(\hat{a},\hat{b})$ are obtained from Equations (2.50) to (2.53).

Finally, to obtain the $100(1-\alpha)\%$ confidence regions for a and b, we use the following approximation ([ROU73])

$$\ell nL(\hat{a},\hat{b}|\underline{y},\underline{t}) - \ell nL(a,b|\underline{y},\underline{t}) = \frac{1}{2}\chi^2_{2;\alpha}$$

or

$$\ell nL(a,b|\underline{y},\underline{t}) = \ell nL(\hat{a},\hat{b}|y,t) - \frac{1}{2}\chi^2_{2;\alpha} \tag{2.58}$$

where $\ell nL(\hat{a},\hat{b}|\underline{y},\underline{t})$ represents the value of the log-likelihood function at $a = \hat{a}$ and $b = \hat{b}$.

Substituting Equation (2.45) in Equation (2.58), we get

$$\sum_{i=1}^{n}(y_i-y_{i-1})\ell na + \sum_{i=1}^{n}(y_i-y_{i-1})\ell n(e^{-bt_{i-1}}-e^{-bt_i})$$

$$-\sum_{i=1}^{n}\ell n\{(y_i-y_{i-1})!\} - a(1-e^{-bt_n}) = C, \quad (2.59)$$

where

$$C = \ell nL(\hat{a},\hat{b}|\underline{y},\underline{t}) - \frac{1}{2}\chi^2_{2;\alpha} . \quad (2.60)$$

Equation (2.59) defines a contour of the $100(1-\alpha)\%$ confidence region. For given data, $\hat{a}$, $\hat{b}$, and α, Equation (2.59) can be solved for those values of a and b which satisfy it. (For computational purposes, it is easier to take values of a ($\geq \hat{a}$) and solve for the corresponding values of b.)

2.5.2 Estimation When Times Between Failures Are Given

Now we consider the case when data is available in the form of times between individual failures. As mentioned earlier, such data is not common and is rarely available.

Recall that $X_1, X_2, \ldots, X_n$ denote the times between failures and $S_n = \sum_{i=1}^{n} X_i$. Then the data is in the form

$\underline{x} = (x_1, x_2, \ldots, x_n)$ and $s_n = \sum_{i=1}^{n} x_i$. The distribution of times between failures was discussed in section 2.4.3 and is obtained from Equations (2.40) and (2.41), as

$$\phi_{S_1,\ldots,S_n}(s_1,\ldots,s_n) = (ab)^n e^{-b\sum_{i=1}^{n} s_i} \cdot e^{-a(1-e^{-bs_n})} .$$

The likelihood function for a, b, given $\underline{s}$, is the same as above and can be written as

$$L(a,b|\underline{s}) = (ab)^n \cdot e^{-b\sum_{i=1}^{n} s_i} \cdot e^{-a(1-e^{-bs_n})} . \quad (2.61)$$

Then the log (natural) likelihood is

$$\ell nL(a,b|\underline{s}) = n\ell na + n\ell nb - b\sum_{i=1}^{n} s_i - a(1-e^{-bs_n}) . \quad (2.62)$$

To get the maximum likelihood estimates $\hat{a}$, $\hat{b}$, we take the partial derivatives of Equation (2.62) and equate them to zero, i.e.,

$$\frac{\partial \ell nL}{\partial a} = 0 , \quad (2.63)$$

and

$$\frac{\partial \ell nL}{\partial b} = 0 \ . \qquad (2.64)$$

These equations yield

$$\frac{n}{a} = 1 - e^{-bs_n} \qquad (2.65)$$

and

$$\frac{n}{b} = as_n \cdot e^{-bs_n} + \sum_{i=1}^{n} s_i \ . \qquad (2.66)$$

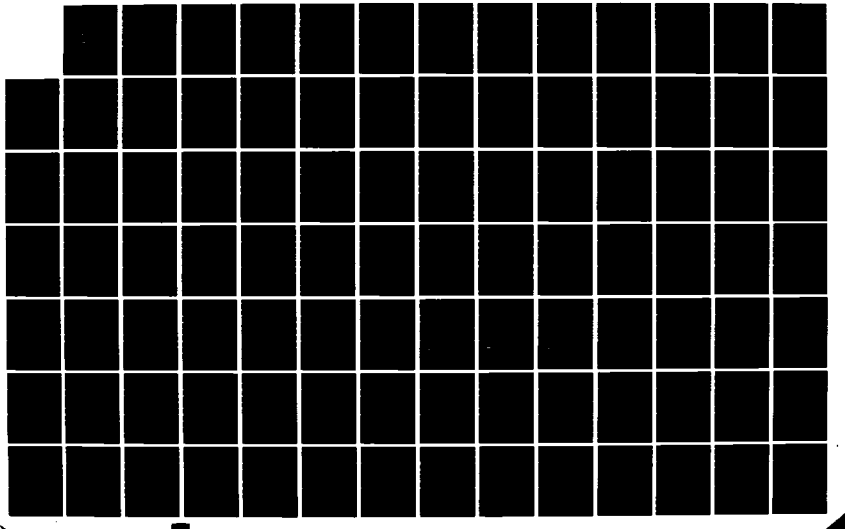
As in the first case, these equations do not yield simple analytical solutions and have to be solved numerically. The solutions of Equations (2.65), and (2.66) are the mle's $\hat{a}$ and $\hat{b}$.

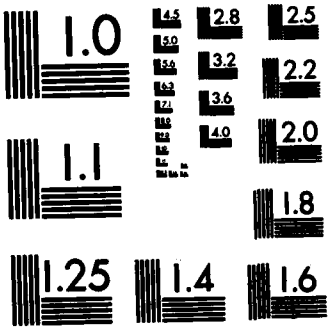
Regarding the asymptotic distribution of $(\hat{a},\hat{b})$, recall that (see section 2.4.2) the joint density of S_1, ...,S_n is improper. Therefore, the asymptotic properties of mle's do not hold in this case.

To obtain the $100(1-\alpha)\%$ confidence regions for (a,b), we use the same approximation as was used in section 2.5.1, viz.

AD-A123 421 SOFTWARE RELIABILITY MODELLING AND ESTIMATION TECHNIQUES(U) SYRACUSE UNIV N Y DEPT OF INDUSTRIAL ENGINEERING AND OPERATIONS RESEARCH A L GOEL OCT 82 2/4

UNCLASSIFIED RADC-TR-82-263 F30602-78-C-0351 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

$$\ell nL(\hat{a},\hat{b}|\underline{s}) - \ell nL(a,b|\underline{s}) = \frac{1}{2}\chi^2_{2;\alpha} \quad . \tag{2.67}$$

From Equations (2.62) and (2.67), a contour of the $100(1-\alpha)\%$ confidence region is obtained as

$$n\ell na + n\ell nb - b\sum_{i=1}^{n} s_i - a(1-e^{-bs_n}) = C, \tag{2.68}$$

where

$$C = \ell nL(\hat{a},\hat{b}|\underline{s}) - \frac{1}{2}\chi^2_{2;\alpha} \quad . \tag{2.69}$$

As before, Equation (2.68) can be solved for given $\underline{s}$, $\hat{a}$, $\hat{b}$, and α to get the desired contours.

2.6 GOODNESS-OF-FIT TEST

In this section, we describe the Kolmogorov-Smirnov goodness-of-fit test (K-S Test) to check whether the NHPP model developed in sections 2.2 and 2.5 provides a good fit to a given set of failure data.

Consider the case when the data are given as a sequence of software failure times $\underline{s} = (s_1, s_2, \ldots, s_n)$. We want to test whether the events $\underline{s}$ are generated from a NHPP. Suppose that $0 \leq S_1 \leq S_2 \leq \ldots \leq S_n$ are the random times at which the first n events occur in a NHPP with unknown mean value function m(t). We wish to test the simple hypothesis

$$H_0: \quad m(t) = m_0(t) \quad \text{for} \quad t \geq 0,$$

versus

$$H_1: \quad m(t) \neq m_0(t) \quad \text{for} \quad t \geq 0.$$

Writing $m_0(t) = a_0(1-e^{-b_0 t})$, the hypothesis H_0 can be written as

$$H_0: \quad m(t) = a_0(1-e^{-b_0 t}) \quad \text{for} \quad t \geq 0. \qquad (2.70)$$

For testing purposes, we need the joint conditional distribution of the failure times. The following theorem is useful in deriving this distribution.

Theorem. Given that $N(t) = n$, the n failure times $0 \leq S_1 \leq S_2 \leq \ldots \leq S_n$ in the interval $[0,t]$ are random variables whose joint conditional distribution is the same as the distribution of the order statistics of a random sample of size n from the distribution $G(x) = \frac{m(x)}{m(t)}$ for $0 \leq x \leq t$.

For proof of this Theorem, see Cox and Lewis [COX66].

Corollary. Given that $S_n = t$, the (n-1) failure times $0 \leq S_1 \leq S_2 \leq \ldots \leq S_{n-1}$ have the same joint conditional distribution as the order statistics of a random sample of size (n-1) from the distribution $G(x) = \frac{m(x)}{m(t)}$.

This Corollary easily follows from the above Theorem. Using this Corollary, we reduce the hypothesis of Equation (2.70) to

$$H_0: \quad G(x) = G_0(x) = \frac{m_0(x)}{m_0(t)} \quad \text{for} \quad 0 \leq x \leq t. \qquad (2.71)$$

For our case we have

$$H_0: \quad G(x) = \frac{1-e^{-b_0 x}}{1-e^{-b_0 t}} \quad \text{for} \quad 0 \leq x \leq t. \qquad (2.72)$$

Note that the expression in Equation (2.72) represents a truncated exponential distribution.

We now consider the Kolmogorov-Smirnov (K-S) goodness-of-fit test [ROM76, ROV73]. Given the values of a random sample of size n-1, $s_1, s_2, \ldots, s_{n-1}$, we define the sample cdf by $H_{n-1}(x) = k/(n-1)$, where k is the number of sample values $\leq x$. Thus, $H_{n-1}(x)$ is a step function which is zero for x less than s_1, has a jump of $1/(n-1)$ at each s_k, and is 1 for x greater than or equal to s_{n-1}. That is,

$$H_{n-1}(x) = \begin{cases} 0 & , \ x < s_1 \\ k/(n-1) & , \ s_{k-1} \leq x < s_k, \quad k=2,3,\ldots,n-1. \\ 1 & , \ x \geq s_{n-1} \end{cases} \qquad (2.73)$$

Since H_{n-1} is a step function and G is monotonically increasing and continuous, it suffices to test the absolute

deviations at the sample points s_k, k = 1,2,...,n-1, and then take the maximum of these (n-1) values. The following procedure is used for calculating the test statistic D. For each k = 1,2,...,n-1, set

$$D_k = \max\{|G_0(s_k) - \frac{k}{n-1}| \ , \ |G_0(s_k) - \frac{k-1}{n-1}|\}.$$

Then set

$$D = \max_k\{D_k\} \ . \qquad (2.74)$$

If the value of D calculated in Equation (2.74) is greater than or equal to the critical value $D_{n-1;\alpha}$, we reject the null hypothesis H_0 that $S_1,S_2,\ldots,S_{n-1}$ follow $G_0(x)$; otherwise we do not reject the null hypothesis. The critical values $D_{n-1;\alpha}$ associated with the K-S test at a level of significance α are available from statistical tables [ROH 76, p. 661].

It should be noted that, if the parameters of $G_0(x)$ are estimated from the sample, the K-S test can be used but will give extremely conservative results. To achieve better results, the level of significance needs to be adjusted. One approach suggested by Allen [ALL 78] is

to test at the 5% level of significance and use the critical value for the 20% level or test at the 1% level and use the critical value for 10% level. We will use this approach in our analyses in later sections.

Another use of the K-S test in our context is in developing confidence limits for the true cdf G(x). For example, if we take a random sample of size (n-1) and use it to construct the sample cdf $H_{n-1}(x)$, then we can be 100(1-α)% confident that the true cdf G(x) does not deviate from $H_{n-1}(x)$ by more than $D_{n-1;\alpha}$. That is, the 100(1-α)% confidence limits for G(x) are given by

$$H_{n-1}(x) - D_{n-1;\alpha} < G(x) < H_{n-1}(x) + D_{n-1;\alpha} \quad . \quad (2.75)$$

These limits are especially useful in the case when the parameters of $G_0(x)$ are to be estimated from the data. For this case, the null hypothesis H_0 will be rejected at a level of significance α if one or more points of $G_0(x)$ fall outside the 100(1-α)% confidence limits given by Equation (2.75). Otherwise, it will not be rejected.

2.7 ANALYSIS OF FAILURE DATA FROM NAVAL TACTICAL DATA SYSTEM (NTDS)

Jelinski and Moranda [JEL 72] first analyzed some software failure data from the U.S. Navy Fleet Computer Programming Center. Since then, this data set has been used by several investigators for model validation purposes. In this section, we analyze the same data set to see how good the NHPP model is in modelling these failures.

The data set was extracted from information about errors in the development of software for the real-time, multi-computer complex which forms the core of the Naval Tactical Data System (NTDS). The NTDS software consisted of some 38 different project schedules. Each module was supposed to follow three stages: the production (or development) phase, the test phase, and the user phase. Many of the "trouble reports" or "software anomaly reports" were generated whenever a system-level symptom of a deficiency was noted by operators or users. A proper trace back to the exact cause in software of this symptom was done by personnel familiar with the entire system. However, Jelinski and Moranda felt that it was better to analyze the data from isolated modules than from the total system, due to the fact that many

of the modules did not evolve in the fashion indicated. One of the larger modules, denoted by A-module, had the desired pattern. The times (in days) between failures for this module are shown in Table 2.1. Twenty-six software faults were found during the production phase and five additional faults during the test phase. The last fault was found on 4 Jan 1971. One fault was observed during the user phase on 20 Sept 1971 and two more faults (4 Oct 1971, 10 Nov 1971) during the test phase. This indicates that a re-work of the module had taken place after the user error was found. A more detailed description of the NTDS software can be found in [JEL72].

Data Analyses

The data in this case is available as times between software failures and hence the method described in section 2.5.2 will be used for estimation of parameters. We consider the first 26 data points of Table 2.1, for which $n = 26$ and $s_{26} = \sum_{k=1}^{26} x_k = 250$ days.

To get an appreciation of the likelihood function associated with this data set, the log-likelihood from Equation (2.62) is plotted in Figure 2.2. We see that

TABLE 2.1

SOFTWARE FAILURE DATA FROM NTDS

ERROR NO.	TIME BETWEEN FAILURES x_k, days	CUMULATIVE TIME $s_n = \Sigma x_k$, days
Production (Checkout) Phase		
1	9	9
2	12	21
3	11	32
4	4	36
5	7	43
6	2	45
7	5	50
8	8	58
9	5	63
10	7	70
11	1	71
12	6	77
13	1	78
14	9	87
15	4	91
16	1	92
17	3	95
18	3	98
19	6	104
20	1	105
21	11	116
22	33	149
23	7	156
24	91	247
25	2	249
26	1	250
Test Phase		
27	87	337
28	47	384
29	12	396
30	9	405
31	135	540
User Phase		
32	258	798
Test Phase		
33	16	814
34	35	849

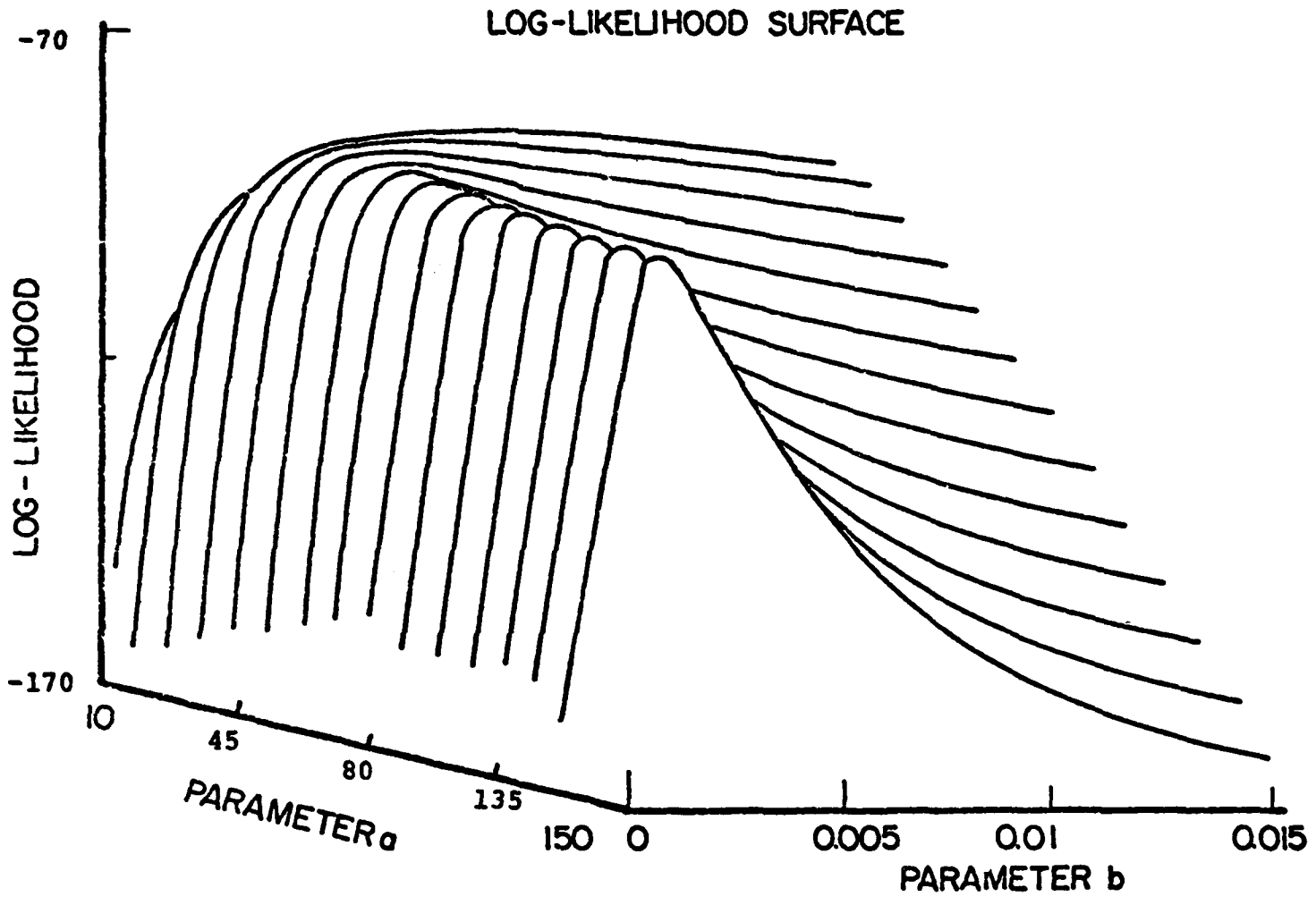


Figure 2.2. Log-Likelihood Surface Based on NTDS Data.

the surface rises sharply along the b-axis and is relatively flat along the a-axis.

The maximum of this surface is obtained by solving Equations (2.65) and (2.66). Substituting the appropriate values from Table 2.6 in Equations (2.65) and (2.66) we get

$$\frac{26}{a} = 1 - e^{-b(250)} \qquad (2.76)$$

and

$$\frac{26}{b} = a(250) \cdot e^{-b(250)} + 250. \qquad (2.77)$$

Solving Equations (2.76) and (2.77) numerically, we get

$$\hat{a} = 33.99$$

and

$$\hat{b} = 0.00579$$

as the mle's for a and b, respectively. The fitted mean value function is

$$\hat{m}(t) = 33.99(1 - e^{-0.00579t}) \quad . \qquad (2.78)$$

and is shown in Figure 2.3, along with the actual data (determination of the confidence bounds will be discussed later).

Goodness-of-fit Test

We now perform the Kolmogorov-Smirnov goodness-of-fit test to check the adequacy of the fitted model. Now, using the Corollary and the results in Section 2.6, we conduct the test based on 26-1 = 25 points. The hypothesis, from Equation (2.71), is

$$H_0: \quad G_0(x) = \frac{1-e^{-b_0 x}}{1-e^{-b_0(250)}} \text{ for } 0 \leq x \leq 250, \qquad (2.79)$$

and the sample cdf is

$$H(x) = \begin{cases} 0 & , x < s_1 \\ k/25 & , s_{k-1} \leq x < s_k, \; k=2,3,\ldots,25 \\ 1 & , x \geq s_{25} \end{cases} \quad . \qquad (2.80)$$

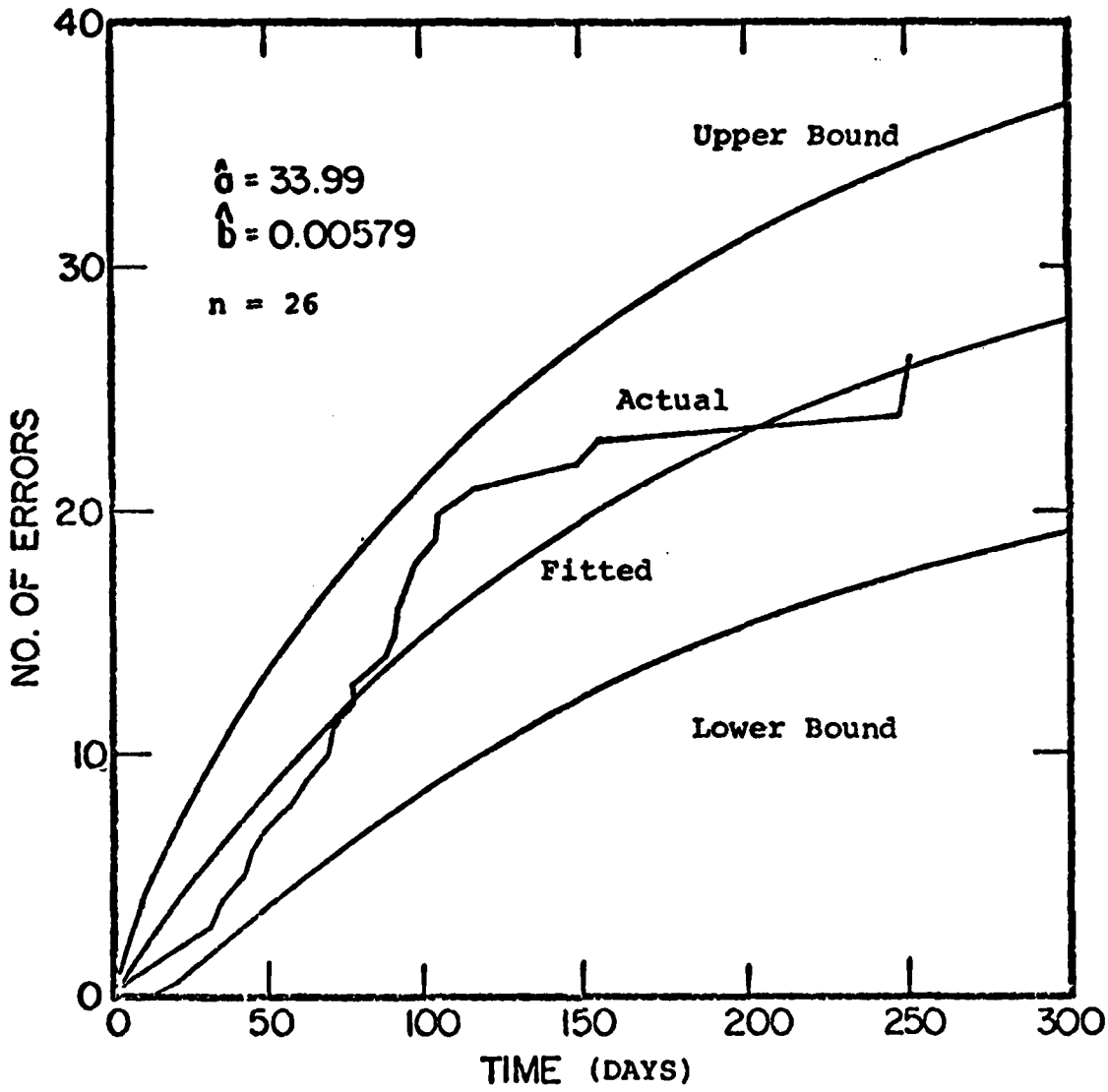


Figure 2.3. Plots of Mean Value Function and 90% Confidence Bounds for the N(t) Process (NTDS Data)

The values of s_k and $H(s_k)$ are given in Table 2.2. To compute $G_0(s_k)$ for various s_k values, we replace b_0 by $\hat{b}$ in Equation (2.79) and obtain Column 4 of Table 2.2. Entries in Columns 5 and 6 are easily obtained from Columns 3 and 4. Now, from Equations (2.47) and (2.79),

$$D = \max_k \{ |G_0(s_k) - H(s_k)|, \ |G_0(s_k) - H(s_{k-1})| \}.$$

In other words, D is the largest entry in Columns 5 and 6 and is seen to be

$$D = 0.2044.$$

To test at $\alpha = .05$, we use a critical value corresponding to $\alpha = .20$ as discussed in section 2.6.

From statistical tables,

$$D_{25;0.2} = 0.208 \ .$$

Since $D < D_{25;0.2}$, we accept the null hypothesis, H_0, at 5% level of significance.

The $100(1-\alpha)\%$ confidence limits for $G(x)$ can now be calculated from Equation (2.75). For example, for

TABLE 2.2

KOLMOGOROV-SMIRNOV TEST
FOR THE NTDS DATA SET

k	s_k	$H(s_k)$	$G_0(s_k)$	$\lvert G_0(s_k)-H(s_k)\rvert$	$\lvert G_0(s_k)-H(s_{k-1})\rvert$
1	9	0.04	0.0664	0.0264	0.0664
2	21	0.08	0.1497	0.0697	0.1097
3	32	0.12	0.2211	0.1011	0.1411
4	36	0.16	0.2460	0.0860	0.1260
5	43	0.20	0.2882	0.0882	0.1282
6	45	0.24	0.2999	0.0599	0.0999
7	50	0.28	0.3286	0.0486	0.0886
8	58	0.32	0.3730	0.0530	0.0930
9	63	0.36	0.3996	0.0396	0.0796
10	70	0.40	0.4357	0.0357	0.0757
11	71	0.44	0.4407	0.0007	0.0407
12	77	0.48	0.4703	0.0097	0.0303
13	78	0.52	0.4751	0.0449	0.0049
14	87	0.56	0.5174	0.0426	0.0026
15	91	0.60	0.5355	0.0645	0.0245
16	92	0.64	0.5399	0.1001	0.0601
17	95	0.68	0.5532	0.1268	0.0868
18	98	0.72	0.5661	0.1539	0.1139
19	104	0.76	0.5915	0.1685	0.1285
20	105	0.80	0.5956	0.2044	0.1644
21	116	0.84	0.6395	0.2005	0.1605
22	149	0.88	0.7557	0.1243	0.0843
23	156	0.92	0.7776	0.1424	0.1024
24	247	0.96	0.9946	0.0346	0.0746
25	249	1.00	0.9982	0.0018	0.0382

$\alpha = 0.05$, we have $D_{25;0.05} = 0.264$, so that the lower and upper confidence bounds are

$$L(x) = \max\{H(x) - 0.264, 0\}$$

and

$$U(x) = \min\{H(x) + 0.264, 1\},$$

where $H(x)$ is given by Equation (2.80). The 95% bounds for $G(x)$, along with $G_0(x)$, are shown in Figure 2.4. We see that the fitted model seems to be adequate.

Having established that the model provides a good fit, various performance measures of interest can be obtained by substituting the estimated values of a and b in the appropriate equations of sections 2.3 and 2.4.

The estimated mean value function, as given in Equation (2.78), is $\hat{m}(t) = 33.99(1-e^{-0.00579t})$. A plot of $\hat{m}(t)$ and the actual number of faults detected during the production period for this case was given in Figure 2.3. Also shown were the 90% confidence bounds for the N(t) process as computed from Equation (2.15).

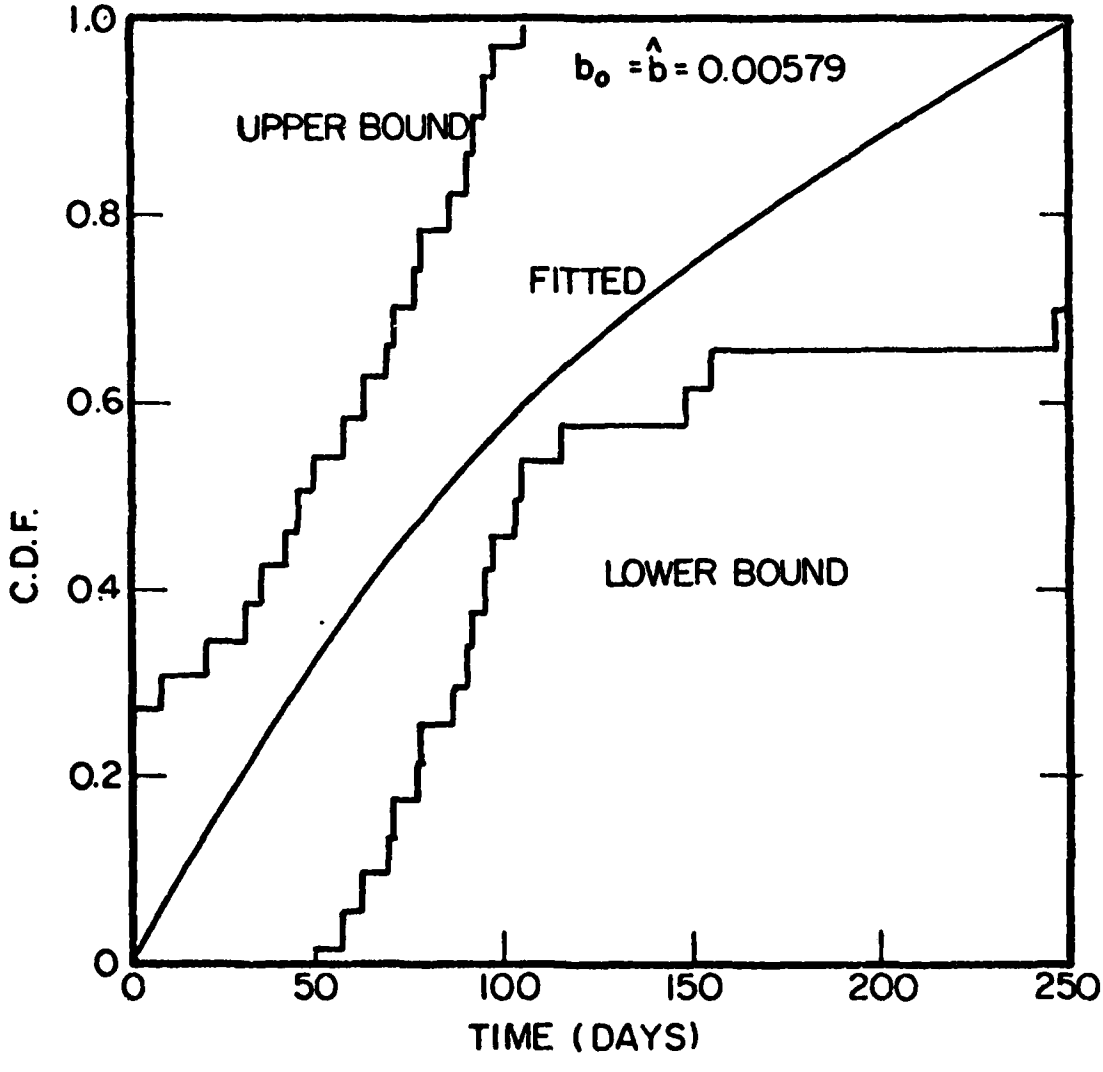


Figure 2.4 95% confidence bounds for the conditional c.d.f. G(x) and the fitted C.D.F. curve (NTDS data)

The $100(1-\alpha)\%$ confidence regions for a and b are obtained from Equations (2.68) and (2.69) following a procedure similar to the one detailed in section 2.7. These are shown in Figure 2.5 for $\alpha = 0.05$, 0.25, and 0.50.

Finally, software reliability, $R_{X_{27}|S_{26}}(x|250)$, can be computed from Equation (2.36). For example, the reliability values after x = 5, 10, 20, and 30 days are 0.796, 0.638, 0.417, and 0.280, respectively. Thus, the probability that the system will operate without any failures for 30 additional days is 0.28. As seen from the data in Table 2.1, the system did operate without any failures for 87 days subsequent to failure number 26.

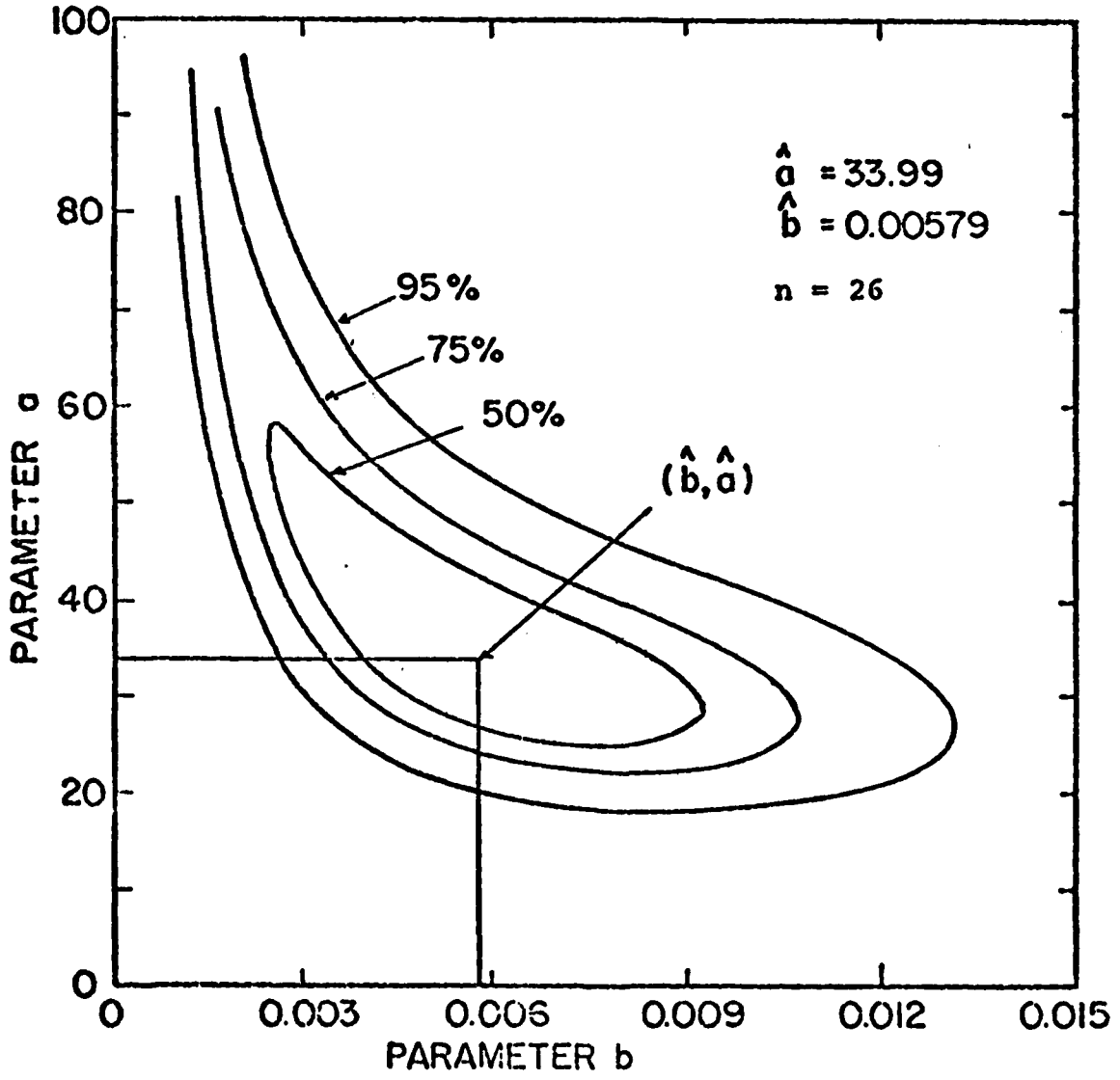


Figure 2.5. Joint Confidence Regions for a and b (NTDS Data)

2.8 ANALYSIS OF FAILURE DATA FROM A LARGE SCALE SOFTWARE SYSTEM

The data to be analyzed in this section have been taken from a large scale project reported in Thayer et al. [THA76]. This project represents an initial delivery of a large command and control software package written in JOVIAL/J4 (JOVIAL is a higher order language generally used for Air Force Command and Control applications). It consists of 115,346 total source statements and 249 routines. Some other characteristics of this project are summarized in Table 2.3. The software was developed functionally, i.e., the project was divided into work units responsible for different functions. Software testing started with developing testing by the development personnel to demonstrate specific functional capabilities, test data extremes, etc.

2.8.1 Failure Data

The failure data used for this study is taken from the Software Problem Reports (SPR's) generated during the formal testing phases of this project. Formal testing, which comprises of validation and acceptance testing, began after development testing. Validation testing was

TABLE 2.3

SOFTWARE PROJECT CHARACTERISTICS

Size (Total source statement)	115,346
Number of routines	249
Language	JOVIAL/J4
Formal Requirements	To function level
Co-contractor	Yes
Subcontractor	No
Operating Mode	Batch
Formal Testing	24 Weeks
Validation	10
Acceptance	2
Integration	10
Operational Demonstration	2

performed by an independent test group at the subsystem level and demonstrated the approved software performance and requirements. Acceptance testing ran a subset of the Validation tests to demonstrate specific requirements. After Acceptance testing, the software underwent final Integration testing by an independent group. Integration testing demonstrated that the applications software correctly interfaced with the operating system and system support software. Finally, Operational Demonstration testing was done to demonstrate the software in an operational environment using an operational timeline and operational data. The data for this error data set was obtained from the four formal test phases (Validation, Acceptance, Integration, and Operational Demonstration) of the applications software. This is so because the majority of the errors analyzed were detected during formal testing.

The time period for the various phases of testing is validation (Jun 1-Aug 12), Acceptance (Aug 13-Aug 24), Integration (Aug 25-Oct 26), and Operational Demonstration (Oct 27-Nov 12) testing. In addition to the above data, operational data spanning a period of approximately nine months was also available and is used for comparison with the predicted values. The only time frame

readily available from the data was the calendar day. The data also contain the mistakes by the operators and the "explanatory" errors, i.e., corrections to make a change to a comment statement or those errors for which a "fix" is not to a routine. These explanatory errors do or do not indicate the type of change. Therefore, the original data was restructured into four sets of data denoted by DS1, DS2, DS3, and DS4 [SUK76]. The description and the total number of faults detected during the formal testing phases for each data set are given in Table 2.4.

In this analysis, the number of software faults detected during formal testing is counted on a weekly basis. Also, for each data set, the software faults detected during the first nine weeks are eliminated due to the fact that we are interested in analyzing the software failures over the period when they are decreasing. The number of SPR's for the 15-week period for the four cases (DS1 to DS4) are given in Table 2.5.

2.8.2 Estimation of Parameters

As seen in Table 2.5, the data for this project are in the form $(t_1,y_1),(t_2,y_2),\ldots,(t_{15},y_{15})$, i.e.,

TABLE 2.4

DESCRIPTION OF THE DATA SETS

DATA SET	DESCRIPTION	TOTAL NUMBER OF FAILURES	
		Formal Testing (24 weeks)	Operation (36 weeks)
DS1	Original Data - TT - EX1 - EX2	2191	198
DS2	Original Data - TT - EX1	2621	263
DS3	Original Data - TT	4367	540
DS4	Original Data - TT - EX2	3937	475

TT represents the mistakes by the operators.

EX1 represents the explanatory errors which do not indicate what type of change (module, documentation, compool, data base) was involved.

EX2 represents the explanatory errors which indicate type of change.

TABLE 2.5

SOFTWARE DATA SETS DS1 TO DS4

WEEK	DATA SET							
	DS1		DS2		DS3		DS4	
	# OF SPR's	CUMULATIVE	# OF SPR's	CUMULATIVE	# OF SPR's	CUMULATIVE	# OF SPR's	CUMULATIVE
1	203	203	238	238	369	369	334	334
2	136	339	179	417	292	667	255	589
3	183	522	222	639	398	1065	359	948
4	47	569	73	712	115	1180	89	1037
5	46	615	56	768	83	1263	73	1110
6	71	686	88	856	150	1413	133	1243
7	54	740	78	934	142	1555	118	1361
8	57	797	92	1026	172	1727	137	1498
9	80	877	104	1130	202	1929	178	1676
10	64	941	85	1215	168	2097	147	1823
11	27	968	53	1268	89	2186	63	1886
12	42	1010	45	1313	87	2273	84	1970
13	55	1065	59	1372	111	2384	107	2077
14	62	1127	84	1456	253	2637	231	2308
15	11	1138	27	1483	70	2707	54	2362

as the number of failures in specified time intervals. Hence, the estimates $\hat{a}$ and $\hat{b}$ are obtained by simultaneously solving Equations (2.47) and (2.48). Thus, by substituting the data set DS1 in Equations (2.47) and (2.48) and solving, we get

$$\hat{a} = 1348, \qquad \hat{b} = 0.124 ,$$

and the fitted mean value function is

$$\hat{m}(t) = 1348(1 - e^{-0.124t}), \quad t \geq 0 .$$

This is also an estimate of the expected number of software failures observed by time t. A plot of the actual cumulative number of failures and the fitted values is given in Figure 2.6.

2.8.3 Goodness-of-fit Test

The goodness-of-fit test is now conducted following the procedure discussed in section 2.6. Since the sample size is 15, the null hypothesis to be tested can be written as

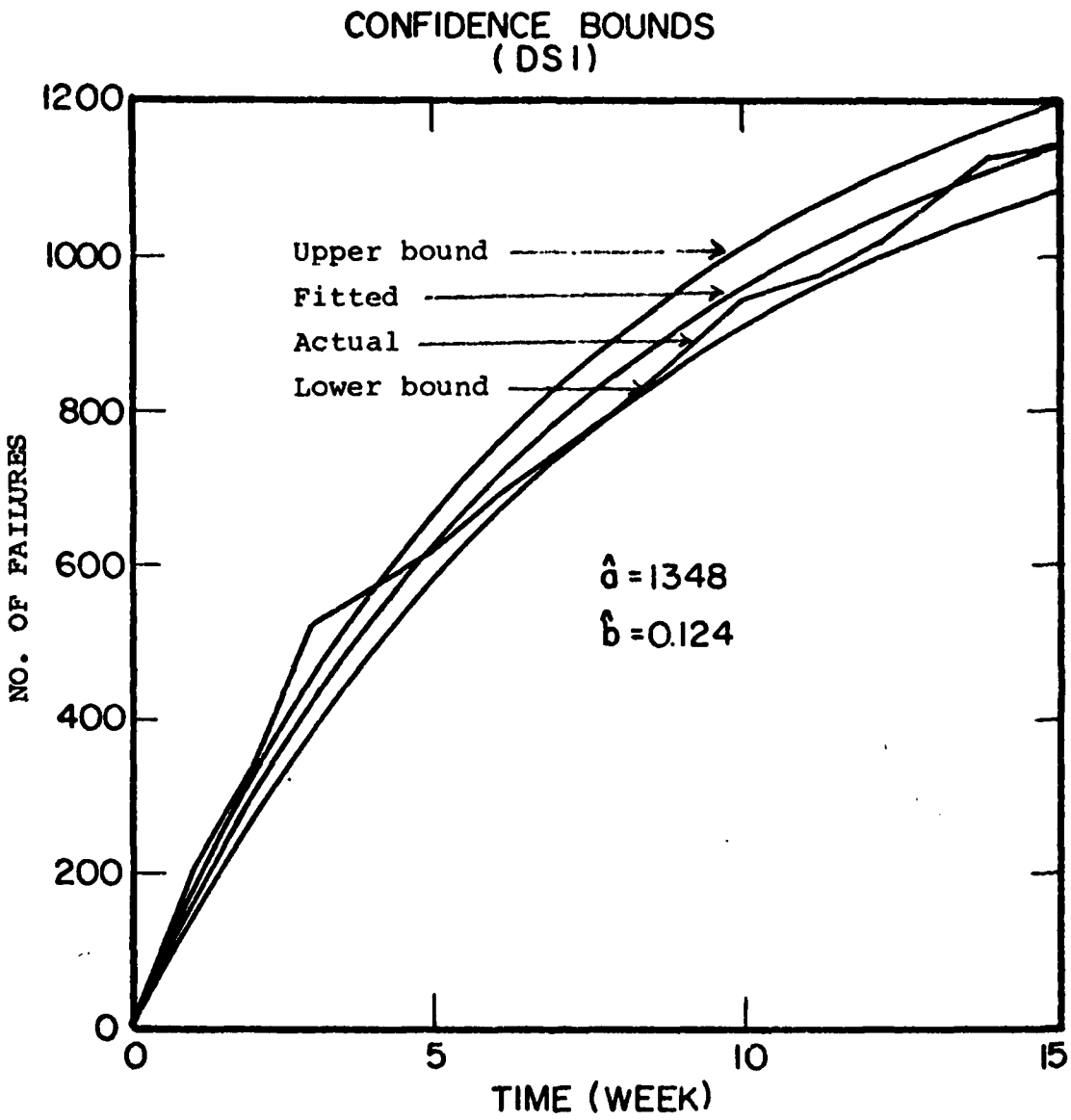


Figure 2.6. Actual and Expected Cumulative Number of Failures and 90% confidence bounds for the N(t) process for data set DS1.

$$H_0: \quad G_0(t_i) = \frac{1-e^{-b_0 t_i}}{1-e^{-b_0(15)}} \quad \text{for} \quad i=1,2,\ldots,15, \quad (2.81)$$

and the sample cdf as

$$H(x) = \begin{cases} 0 & , \ x < t_1 \\ y_i/y_{15} & , \ t_{i-1} < x < t_i, \quad i=2,3,\ldots,15. \\ 1 & , \ x \geq t_{15} \end{cases} \quad (2.82)$$

The computed values of $H(x)$ for various t_i are given in column 2 of Table 2.6.

Now we substitute $b_0 = \hat{b} = 0.124$ in Equation (2.81) and compute the value of $G_0(t_i)$ for $i = 1,2,\ldots,15$. These values are given in column 3 of Table 2.6. Columns 4 and 5 of this table are the quantities needed to find $D = \max_k\{D_k\}$ (see Equation (2.74)). From these columns we find the value of D to be 0.096 corresponding to $t_i = 9$.

To find the critical value corresponding to sample size 15 and $\alpha = .05$, we first note that the parameters had to be estimated in this case. As mentioned in section 2.6, for a situation like this, a suggested approach is to take $\alpha = .20$ to get good results. From

TABLE 2.6

DATA FOR KOLMOGOROV-SMIRNOV TEST
(DATA SET DS1)

t_i	$H(t_i)$	$G_0(t_i)$	$\lvert G_0(t_i)-H(t_i)\rvert$	$\lvert G_0(t_i)-H(t_{i-1})\rvert$
1	0.1784	0.1381	0.0403	0.1381
2	0.2979	0.2601	0.0378	0.0817
3	0.4587	0.3679	0.0908	0.0700
4	0.5000	0.4631	0.0369	0.0044
5	0.5404	0.5472	0.0068	0.0472
6	0.6028	0.6215	0.0187	0.0811
7	0.6503	0.6872	0.0369	0.0844
8	0.7004	0.7452	0.0448	0.0949
9	0.7707	0.7964	0.0257	0.096
10	0.8269	0.8416	0.0147	0.0709
11	0.8506	0.8816	0.031	0.0547
12	0.8875	0.9169	0.0294	0.0663
13	0.9359	0.9481	0.0122	0.0606
14	0.9903	0.9757	0.0146	0.0398
15	1.0000	1.0000	0.0000	0.0097

the statistical tables [ROH76, p. 661], $D_{15;.20} = 0.266$. The observed value D = 0.096 is less than the critical value 0.266 and hence we accept the null hypotheses of Equation (2.76). Thus we conclude that at 5% level of significance the model

$$P\{N(t)=y\} = \frac{\{1348(1-e^{-0.124t})\}}{y!}\{e^{-1348(1-e^{-0.124t})}\}$$

can be considered to provide an adequate fit to data set DS1.

To further check the adequacy of fit, we compute 95% confidence bounds on $G(t_i)$. From Equation (2.75), these bounds are given by

$$H(t_i) - D_{15;.05} < G(t_i) < H(t_i) + D_{15;.05}.$$

From the statistical tables, $D_{15;.05} = 0.366$ and hence the 95% confidence bounds are given by $H(t_i) \pm 0.366$. A plot of these bounds and the fitted values are shown in Figure 2.7.

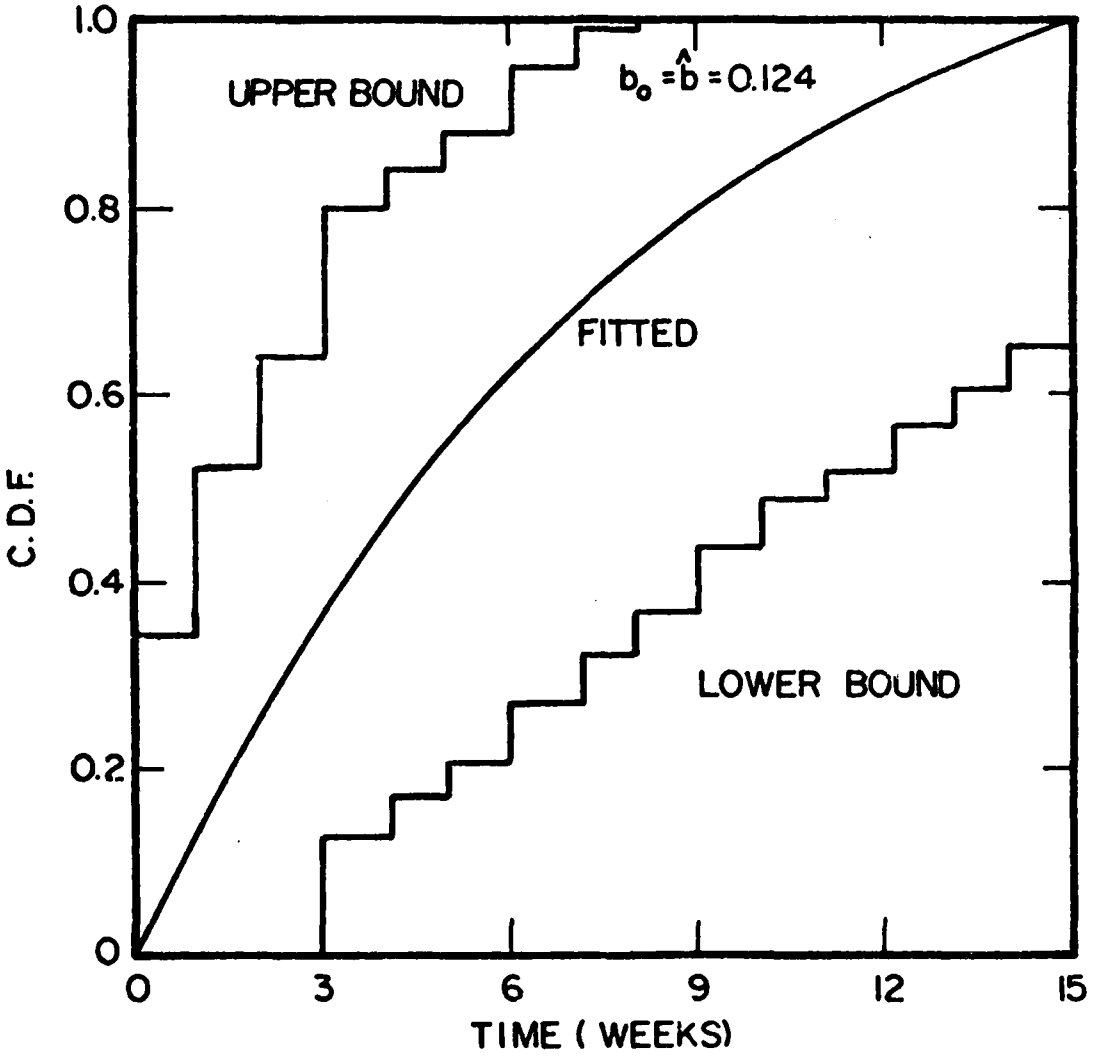


Figure 2.7. 95% confidence bounds for the conditional c.d.f. $G(t_i)$ and the fitted curve for DS1 data

2.8.4 Confidence Regions for (a,b)

To get an appreciation of the variability in the estimated values of a and b, we now construct confidence regions for (a,b). Such regions are given by Equations (2.59) and (2.60). For $\alpha = .05$, the 95% joint confidence region will be the solution of the following equation:

$$\ell nL(a,b|\underline{y},\underline{t}) = \ell nL(a,b|\underline{y},\underline{t}) - \frac{1}{2}\chi^2_{2;.05} ,$$

where

$$\ell nL(\hat{a},\hat{b}|\underline{y},\underline{t}) = \sum_{i=1}^{15}(y_i-y_{i-1})\ell n(1348) + \sum_{i=1}^{15}(y_i-y_{i-1}) \cdot \ell n(e^{-.124t_{i-1}}-e^{-.124t_i}) - \sum_{i=1}^{15}\ell n\{(y_i-y_{i-1})!\} - 1348(1-e^{-.124t_{15}}) .$$

Data $(y_1,t_1),(y_2,t_2),\ldots,(y_{15},t_{15})$ were given in Table 2.5 and

$$\chi^2_{2;.05} = 0.103 .$$

A plot of this region is shown in Figure 2.8. From this plot we see that, even though the most likely values of a and b, based on the data, are $\hat{a} = 1348$, $\hat{b} = 0.124$, the true values can vary over the entire region contained in the 95% contour. Values a = 1450, b = 0.11 will be acceptable (with 95% confidence) and so will a = 1250, b = 0.14. 50% and 75% confidence regions are also shown in Figure 2.8 and can be similarly interpreted.

2.8.5 Variance-Covariance Matrix for $(\hat{a},\hat{b})$

The variance-covariance matrix is useful in quantifying the variability in the estimated parameters and is obtained from Equations (2.50), (2.54), (2.55), and (2.56) by substituting $a = \hat{a} = 1348$, $b = \hat{b} = 0.124$, and the actual data values from Table 2.5. For data set DS1, we get

$$\Sigma_{cov} = \begin{pmatrix} 2368 & -0.2071 \\ -0.2071 & 5.554 \times 10^{-5} \end{pmatrix} .$$

From this we have

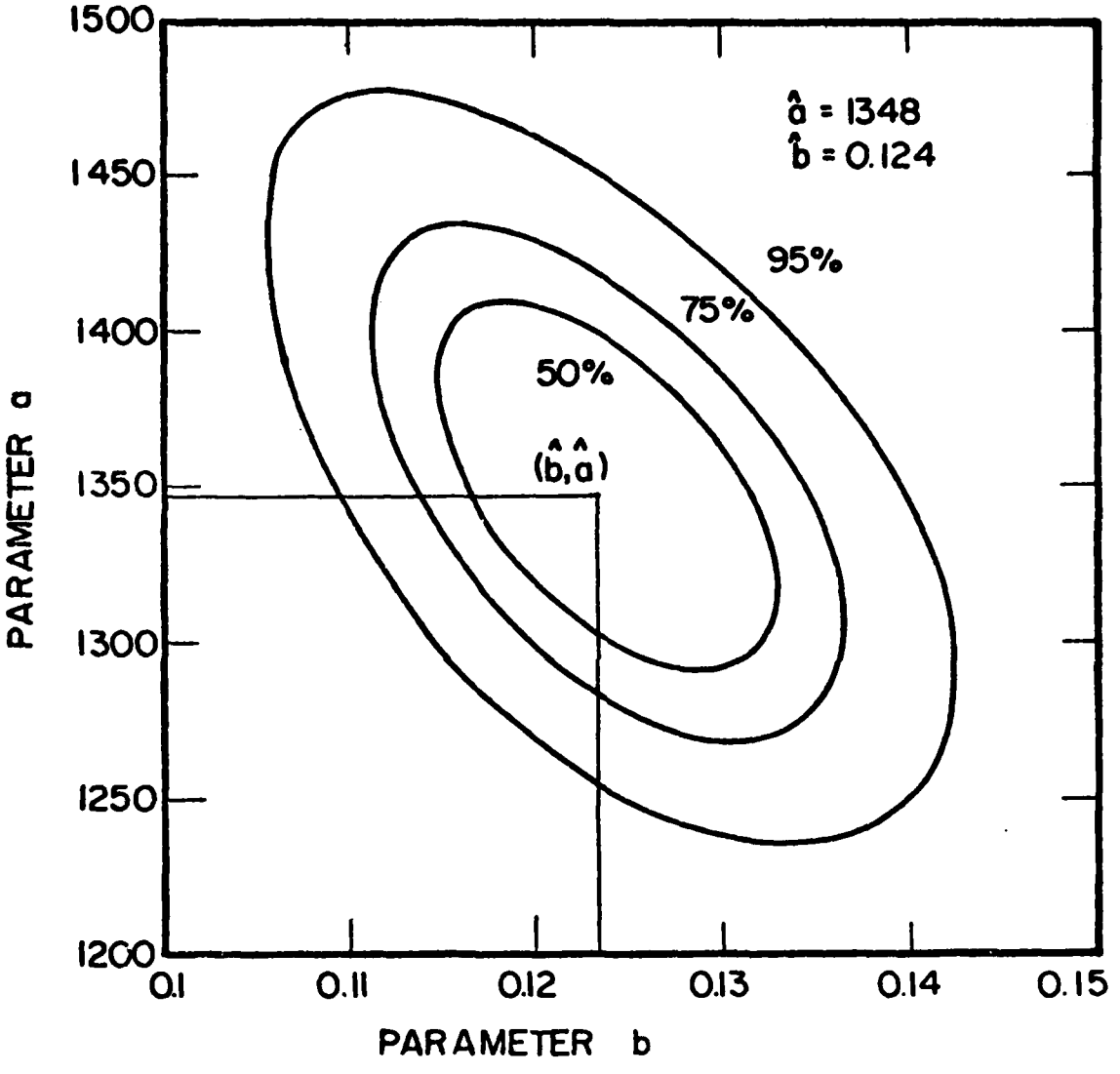


Figure 2.8. Joint confidence regions for a and b for Data Set DS1.

$$\text{Standard Deviation } (\hat{a}) \equiv \sqrt{\text{Var } (\hat{a})} = 48.66$$

$$\text{Standard Deviation } (\hat{b}) \equiv \sqrt{\text{Var } (\hat{b})} = 0.00745$$

$$\text{Correlation Coefficient } (\hat{a},\hat{b}) \equiv \hat{\rho}_{\hat{a},\hat{b}}$$

$$= \frac{-0.2071}{\sqrt{(2368)(5.554 \times 10^{-5})}} = -0.571 \ .$$

2.8.6 Number of Remaining Errors

One useful quantity is the estimated number of remaining faults or errors in the system after some time t. This value is obtained from Equation (2.19) as

$$E\{\overline{N}(t)\} = \hat{a} \cdot e^{-\hat{b}t}$$

or

$$E\{\overline{N}(t)\} = 1348e^{-0.124t} \ .$$

A plot of this quantity is shown in Figure 2.9. As expected, this value decreases with time. Also shown is a plot of the "actual" number of remaining errors

which is based on the assumption that all the errors were detected during 36 weeks of operation. It should be noted that this assumption is made for illustration purposes only and, in general, this may not be the case.

It would also be interesting to compute confidence bounds on $E\overline{N}(t)$. Such bounds can be easily computed as follows.

Let $f(a,b)$ denote $E\overline{N}(t)$. Then, it is well known [ROH76, ROU73] that $100(1-\alpha)\%$ confidence bounds for $f(a,b)$ are given by

$$\{\hat{f}(a,b) \pm t_{n-2;\alpha}\sqrt{\hat{V}(\hat{f}(a,b))}\}\ , \qquad (2.83)$$

where

$$\hat{V}(\hat{f}(a,b)) = \left(\frac{\partial f}{\partial a}\ \frac{\partial f}{\partial b}\right)\Sigma\,\mathrm{cov}\left(\begin{matrix}\frac{\partial f}{\partial a}\\ \frac{\partial f}{\partial b}\end{matrix}\right)\Bigg|_{a=\hat{a},\ b=\hat{b}} \qquad (2.84)$$

and $t_{n-2;\alpha/2}$ is the upper $100(\alpha/2)$ percentage point of the t-distribution with $(n-2)$ degrees of freedom.

The 90% confidence limits for $E\{\overline{N}(t)\}$ for data set DS1 are computed from the above equations and are plotted in Figure 2.9.

2.8.7 Software Reliability

As mentioned in Section 2.4, software reliability is a commonly used performance measure to assess how reliable the system is at various times. To compute software reliability, we use Equation (2.36) and get

$$\hat{R}_{X_k|S_{k-1}}(x|s) = e^{-\hat{a}(e^{-\hat{b}s}-e^{-\hat{b}(s+x)})} .$$

This gives the reliability after time x starting from the current time s. For example, starting from s = 15, the reliability after 0.04 weeks, i.e., at s+x = 15.04, is

$$\hat{R}(0.04|s=15) = e^{-1348(e^{-(.124)15}-e^{-(.124)(15.04)})}$$

or

$$\hat{R}(15.04) = 0.354.$$

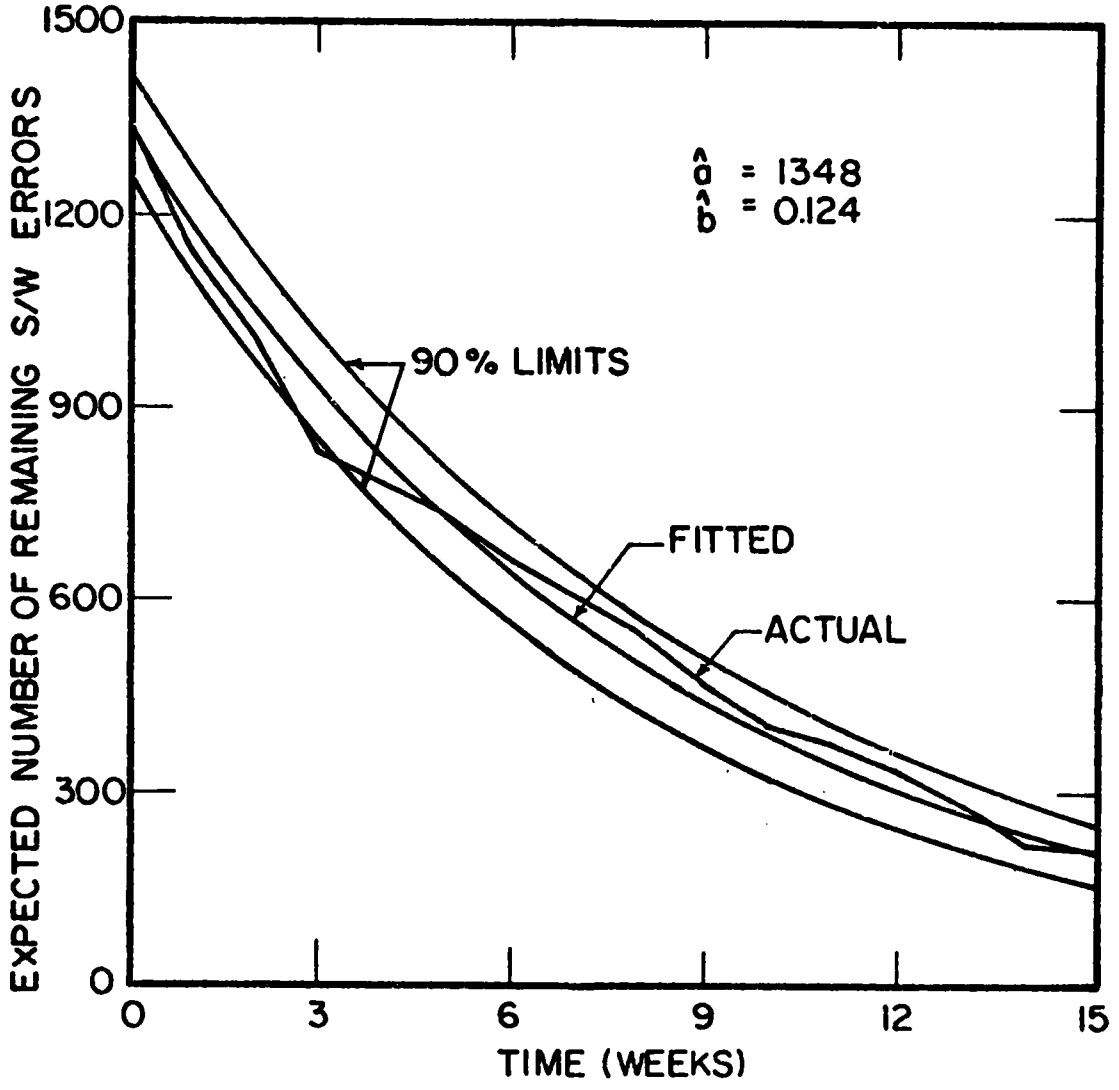


Figure 2.9. Expected number of remaining software errors and related quantities for various t (Data Set DS1)

To see how reliability varies with time, a plot of $\hat{R}(x|s=15)$ is shown in Figure 2.10.

To obtain confidence bounds on reliability, we use a procedure similar to the one used for getting bounds on $E\{\bar{N}(t)\}$. Let $\hat{g}(a,b)$ represent $\hat{R}(x|s=15)$. Then the confidence bounds are given by

$$\{\hat{g}(a,b) \pm t_{n-2;\alpha/2}\sqrt{\hat{V}(\hat{g}(a,b))}\}\ , \tag{2.85}$$

where

$$\hat{V}(\hat{g}(a,b)) = \left(\frac{\partial g}{\partial a}\ \frac{\partial g}{\partial b}\right)\Sigma_{cov}\left(\begin{matrix}\frac{\partial g}{\partial a}\\ \frac{\partial g}{\partial b}\end{matrix}\right)\Bigg|_{\substack{a=\hat{a}\\ b=\hat{b}}} \tag{2.86}$$

90% confidence bounds computed from these equations for the given data are shown in Figure 2.10.

Analyses similar to those for data set DS1 were undertaken for data sets DS2, DS3, and DS4 of Table 2.5. A summary of the results is given in Table 2.7.

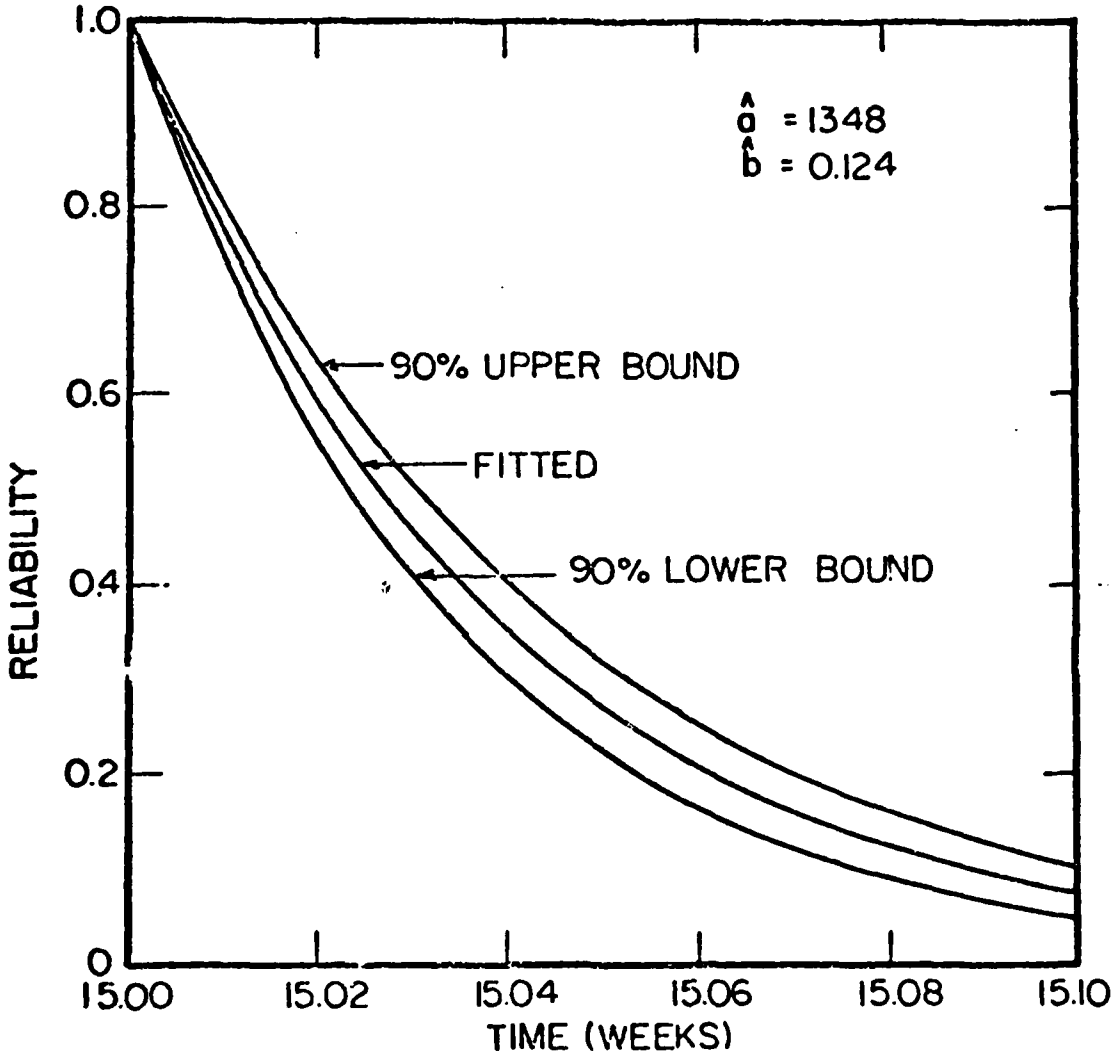


Figure 2.10. Reliability and 90% confidence bounds after 15 weeks of testing

TABLE 2.7

A SUMMARY OF DATA ANALYSES

Quantity \ Data Set	DS1	DS2	DS3	DS4
$\hat{a}$	1348	1823	3958	3446
$\hat{b}$	0.124	0.112	0.0768	0.0771
$\sqrt{\widehat{Var}(\hat{a})}$	48.7	62.2	147.3	136.6
$\sqrt{\widehat{Var}(\hat{b})}$	0.00745	0.00643	0.00460	0.00492
$\hat{\rho}_{a,b}$	-0.571	-0.648	-0.856	-0.855
Estimated Number of Remaining Errors at the end of Operational Demonstration	209	338	1212	1050
Number of Errors Detected During Nine Months of Operation	198	263	540	475

2.9 ANALYSIS OF FAILURE DATA FROM COMMAND AND CONTROL SYSTEMS

In this section, we analyze software failure data from two real-time command and control systems, SYS1 and SYS2. These data sets were reported in [MUS80] and represent failures observed during the system test phase. The number of delivered object instructions for SYS1 was 21,700 and for SYS2, 27,700. The number of programmers for SYS1 and SYS2 was 9 and 5, respectively.

For the first system, a total of 136 failures were observed over 25 hours of execution time and for the second system, the number of failures was 54 over 31 hours of execution time. The observed number of failures per execution hour and the cumulative failures are given in Table 2.8. The number of failures per hour are plotted in Figures 2.11 and 2.12, respectively. The parameters a and b were estimated using Equations (2.65) and (2.66) of Section 2.5 and are

SYS1 $\quad \hat{a} = 142.32 \qquad \hat{b} = 0.125$

SYS2 $\quad \hat{a} = 56.81 \qquad \hat{b} = 0.097$

TABLE 2.8

FAILURES IN ONE HOUR (EXECUTION TIME) INTERVALS
AND CUMULATIVE FAILURES

Hour	SYS1		SYS2	
	No.	Cum.	No.	Cum.
1	27	27	10	10
2	16	43	6	16
3	11	54	4	20
4	10	64	5	25
5	11	75	2	27
6	7	82	1	28
7	2	84	1	29
8	5	89	1	30
9	3	92	0	30
10	1	93	1	31
11	4	97	3	34
12	7	104	7	41
13	2	106	1	42
14	5	111	0	42
15	5	116	0	42
16	6	122	0	42
17	0	122	0	42
18	5	127	4	46
19	1	128	0	46
20	1	129	1	47
21	2	131	1	48
22	1	132	0	48
23	2	134	1	49
24	1	135	1	50
25	1	136	1	51
26			0	51
27			1	52
28			0	52
29			1	53
30			0	53
31			1	54

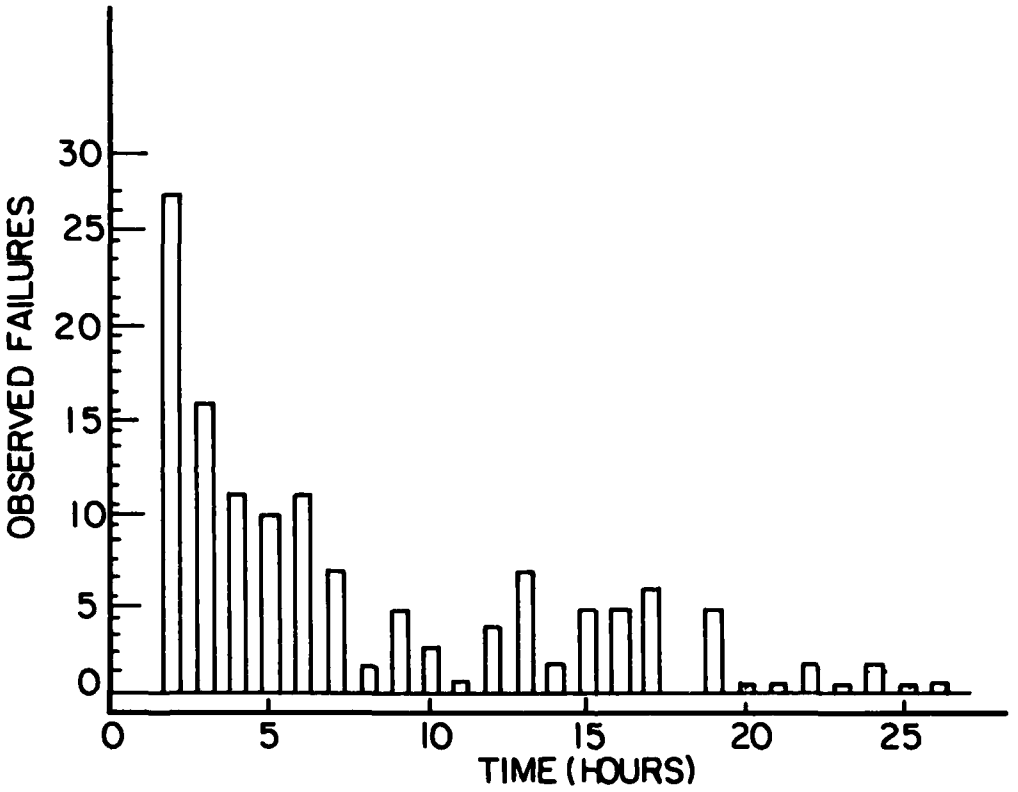


FIG. 2.11 PLOT OF THE NUMBER OF FAILURES PER HOUR (SYS I)

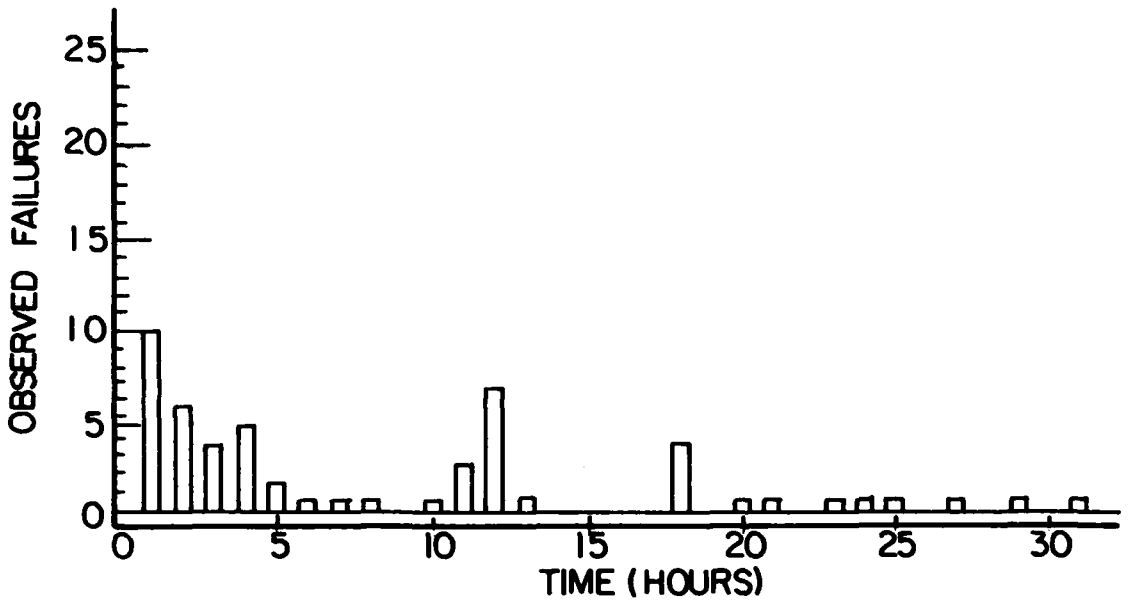


FIG. 2.12 PLOT OF THE NUMBER OF FAILURES PER HOURS (SYS 2)

The fitted models for the mean value function are:

SYS1 $\quad \hat{m}(t) = 142.32(1 - e^{-0.125t})$

SYS2 $\quad \hat{m}(t) = 56.81(1 - e^{-0.097t})$

Plots of the observed cumulative failures and expected failures ($\hat{m}(t)$) are shown in Figures 2.13 and 2.14 for SYS1 and SYS2, respectively.

Observed number of remaining errors and expected number of remaining errors were computed from ($\hat{a}$-N(t)) and $\hat{a} \cdot e^{-\hat{b}t}$, respectively, and are plotted in Figures 2.15 and 2.16 for SYS1 and SYS2, respectively. The 90% confidence bounds for $\hat{m}(t)$ and $\hat{E}(\overline{N}(t))$ are also in Figures 2.13 to 2.16. From a study of these plots, it appears that the fitted models fit the data very well.

Expressions for software reliability for the two systems are obtained from Equation (2.36) as

$$R(x|s=25) = e^{-142.32\{e^{-.125(25)} - e^{-.125(25+x)}\}}$$

and

$$R(x|s=31) = e^{-56.81\{e^{-.097(31)} - e^{-.097(31+x)}\}}.$$

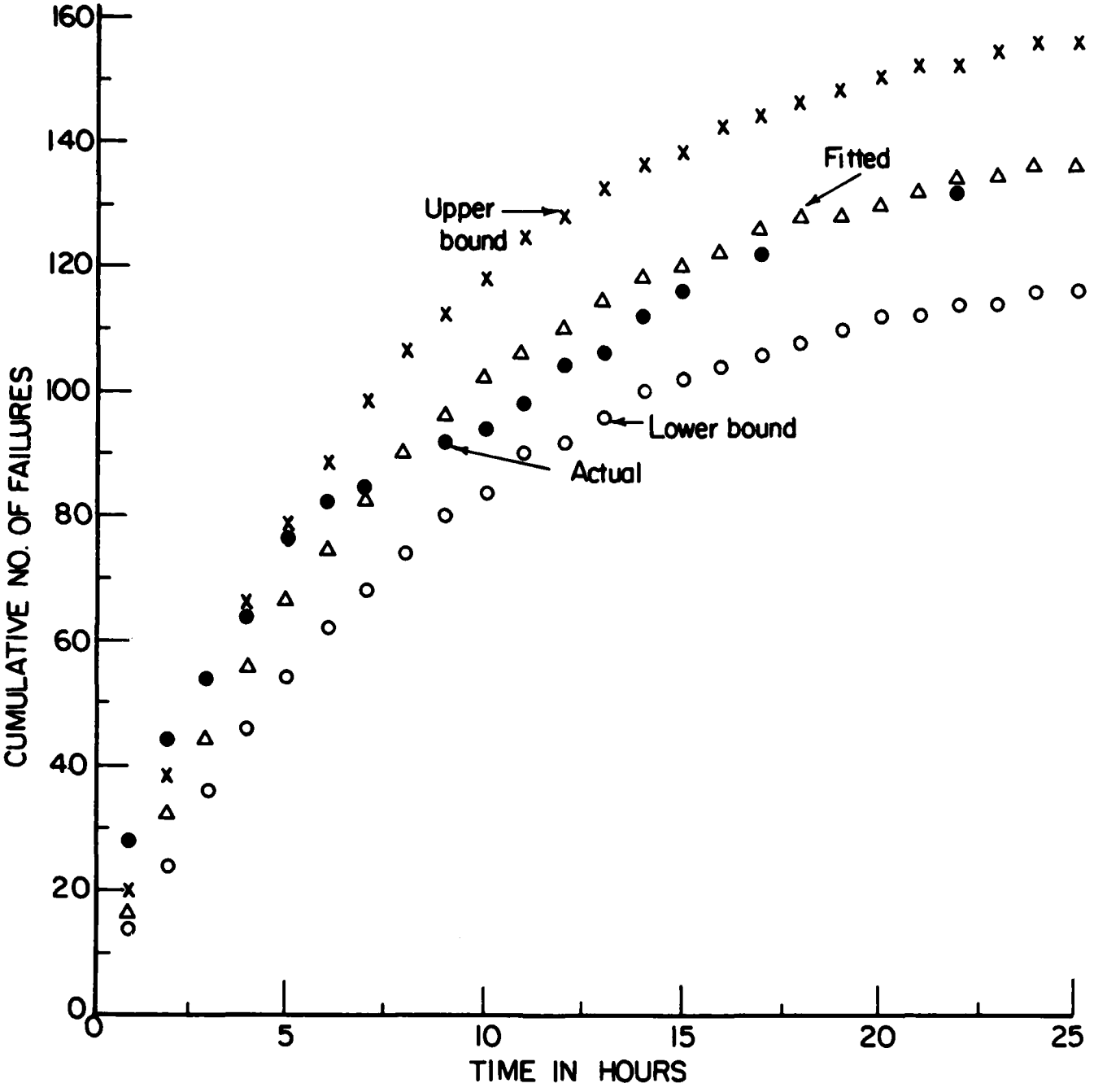


FIG. 2.13 NUMBER OF FAILURES AND 90% CONFIDENCE BOUNDS (SYS 1)

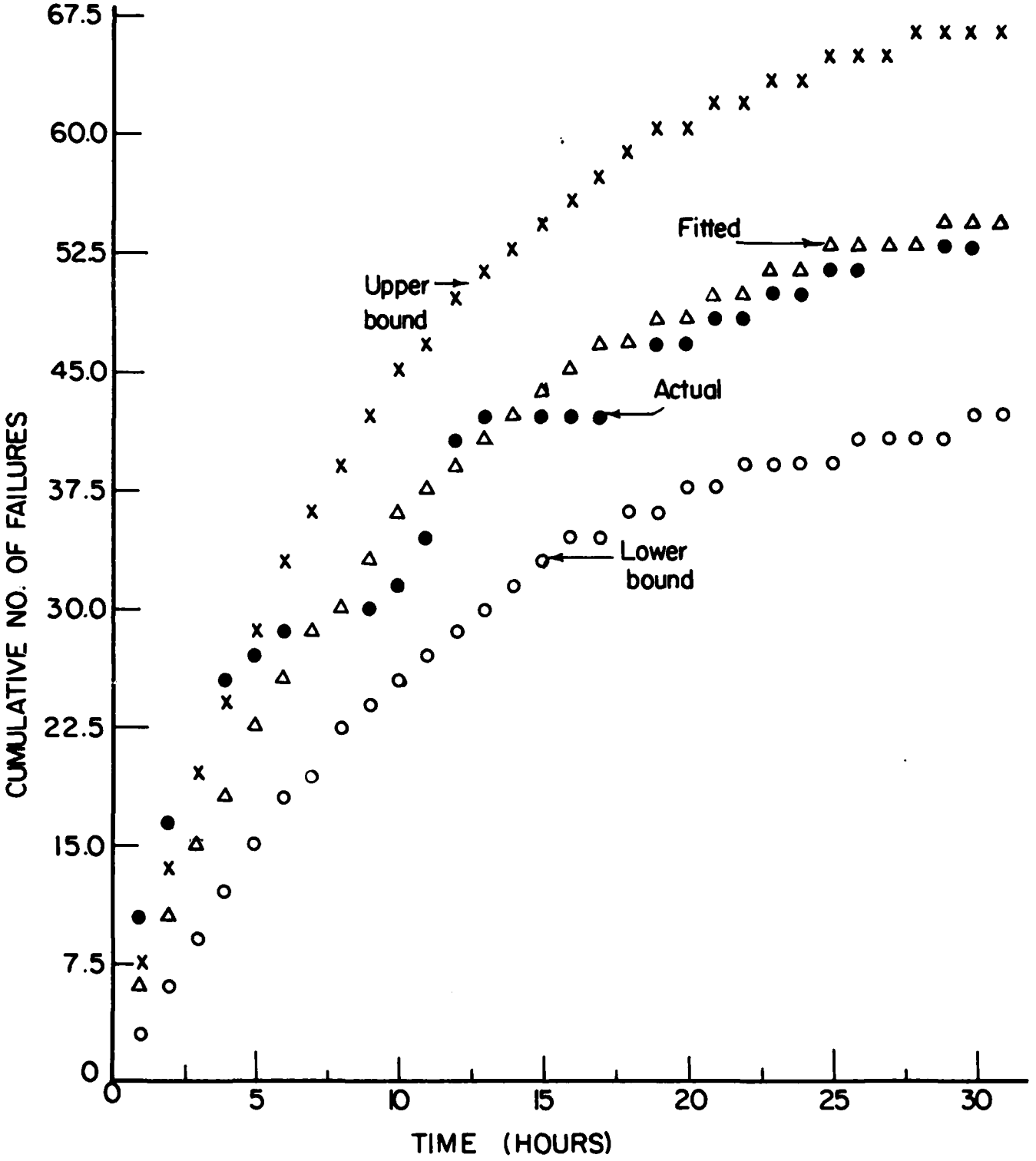


FIG. 2.14 NUMBER OF FAILURES AND 90% CONFIDENCE BOUNDS (SYS 2)

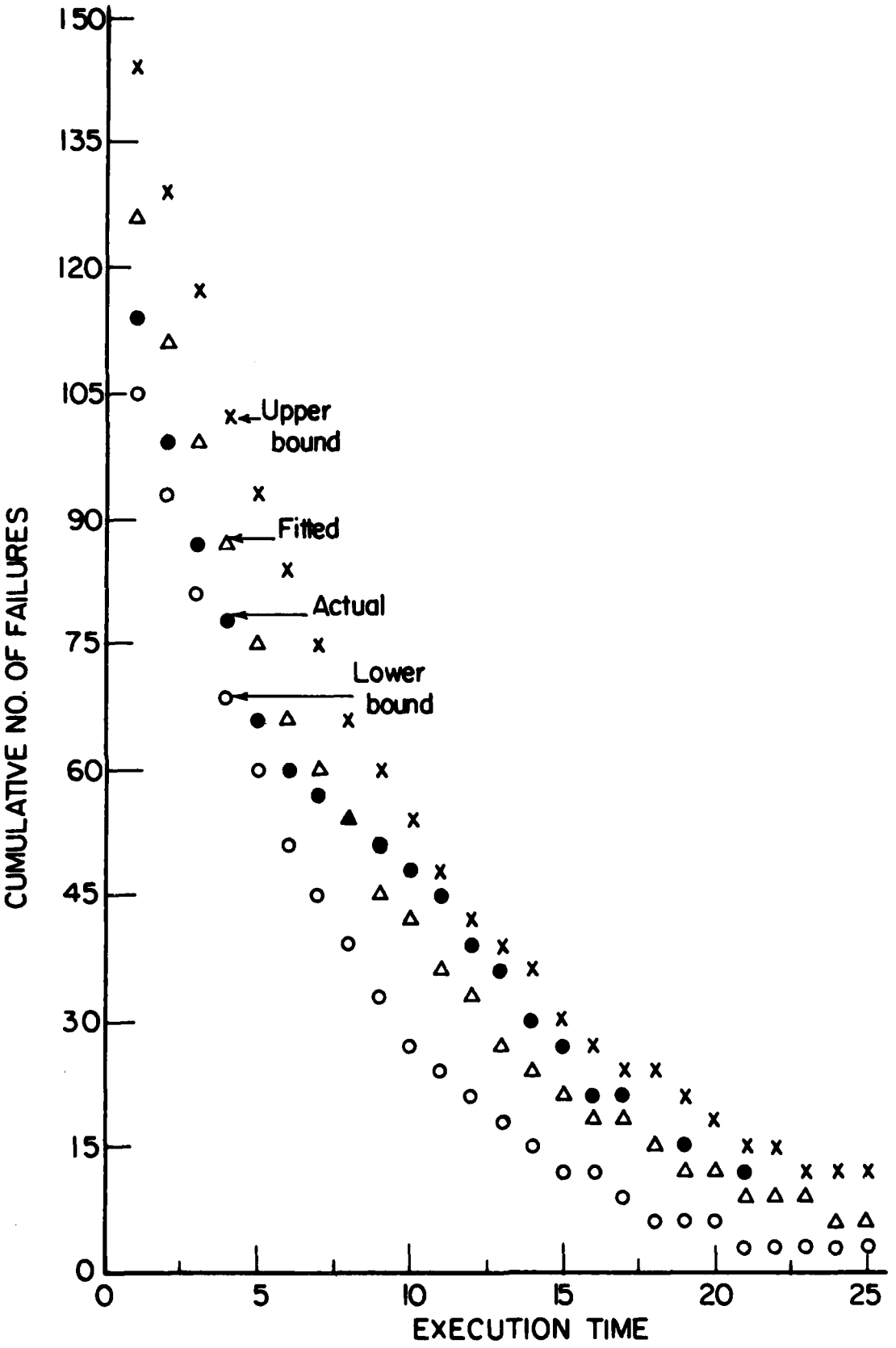


FIG. 2.15 OBSERVED AND EXPECTED NO. OF REMAINING ERRORS WITH 90% CONFIDENCE BOUNDS ON $E[\bar{N}(x)] - SYS$

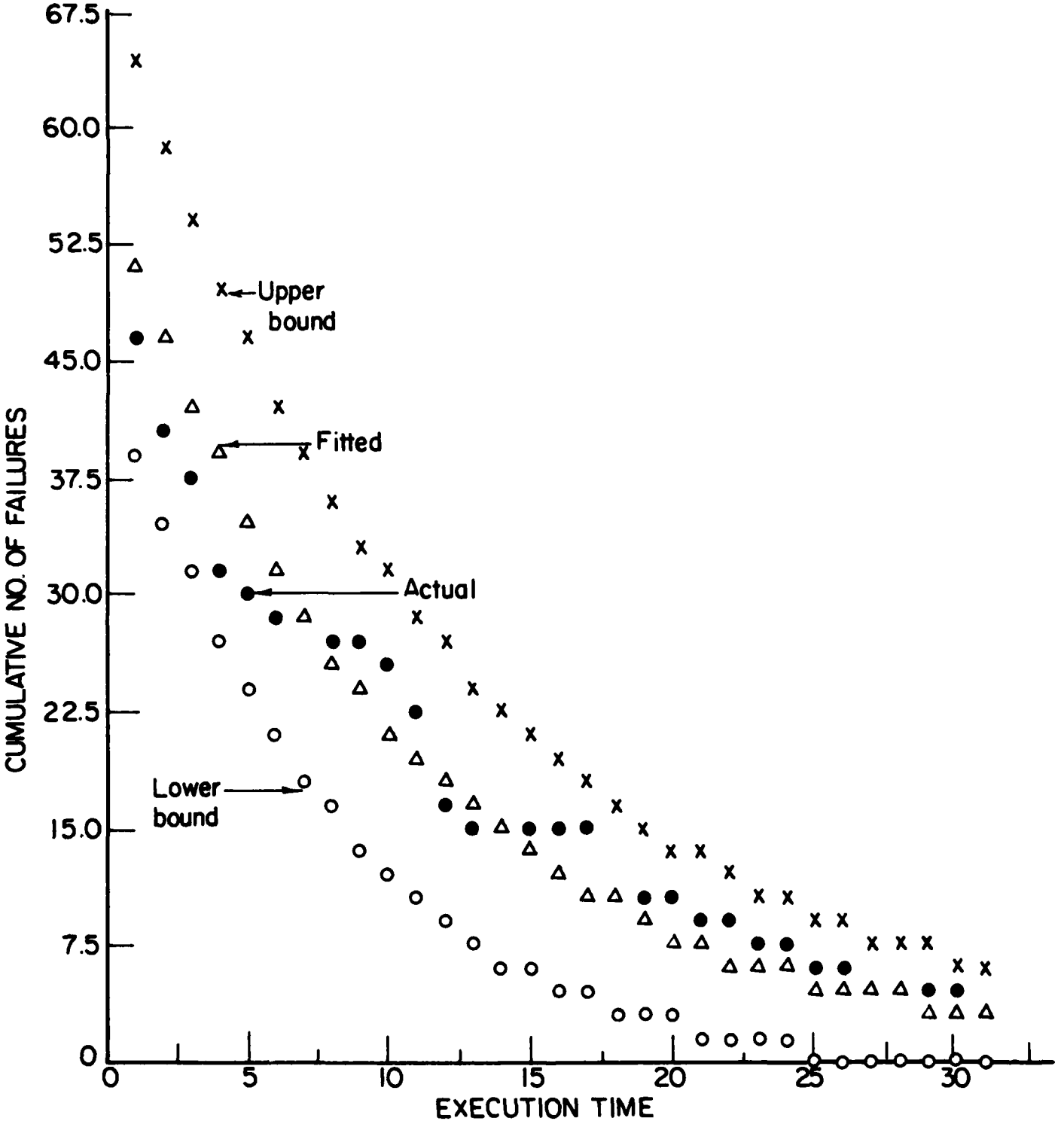


FIG. 2.16 OBSERVED AND EXPECTED NUMBER OF REMAINING ERROR AND 90% CONFIDENCE BOUNDS ON $E[\bar{N}(x)]$ - SYS 2.

Plots of these reliability functions for SYS1 and SYS2, along with 90% confidence bounds, are given in Figures 2.17 and 2.18, respectively.

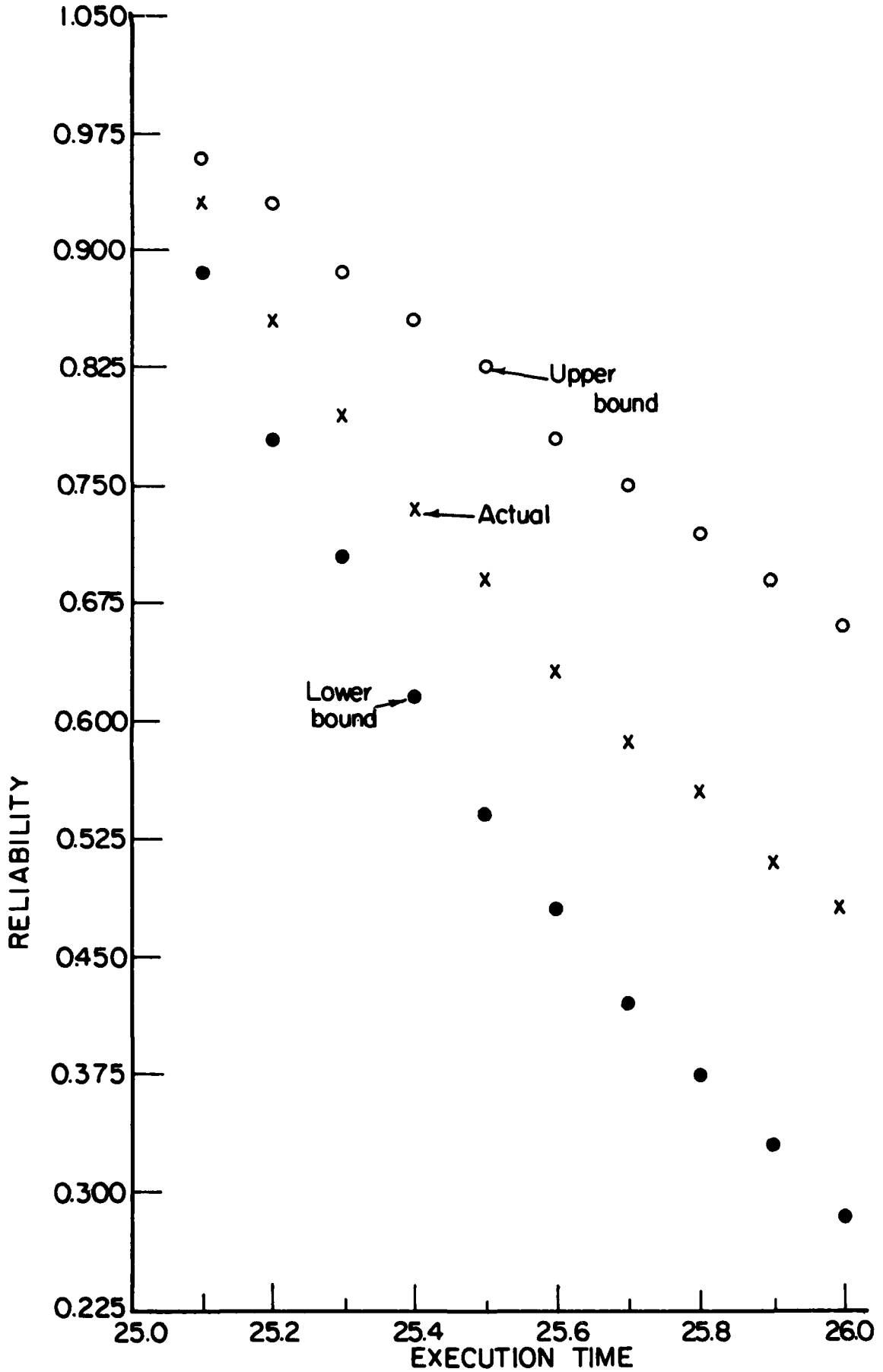


FIG. 2.17 RELIABILITY AND 90% CONFIDENCE BOUNDS - SYS I

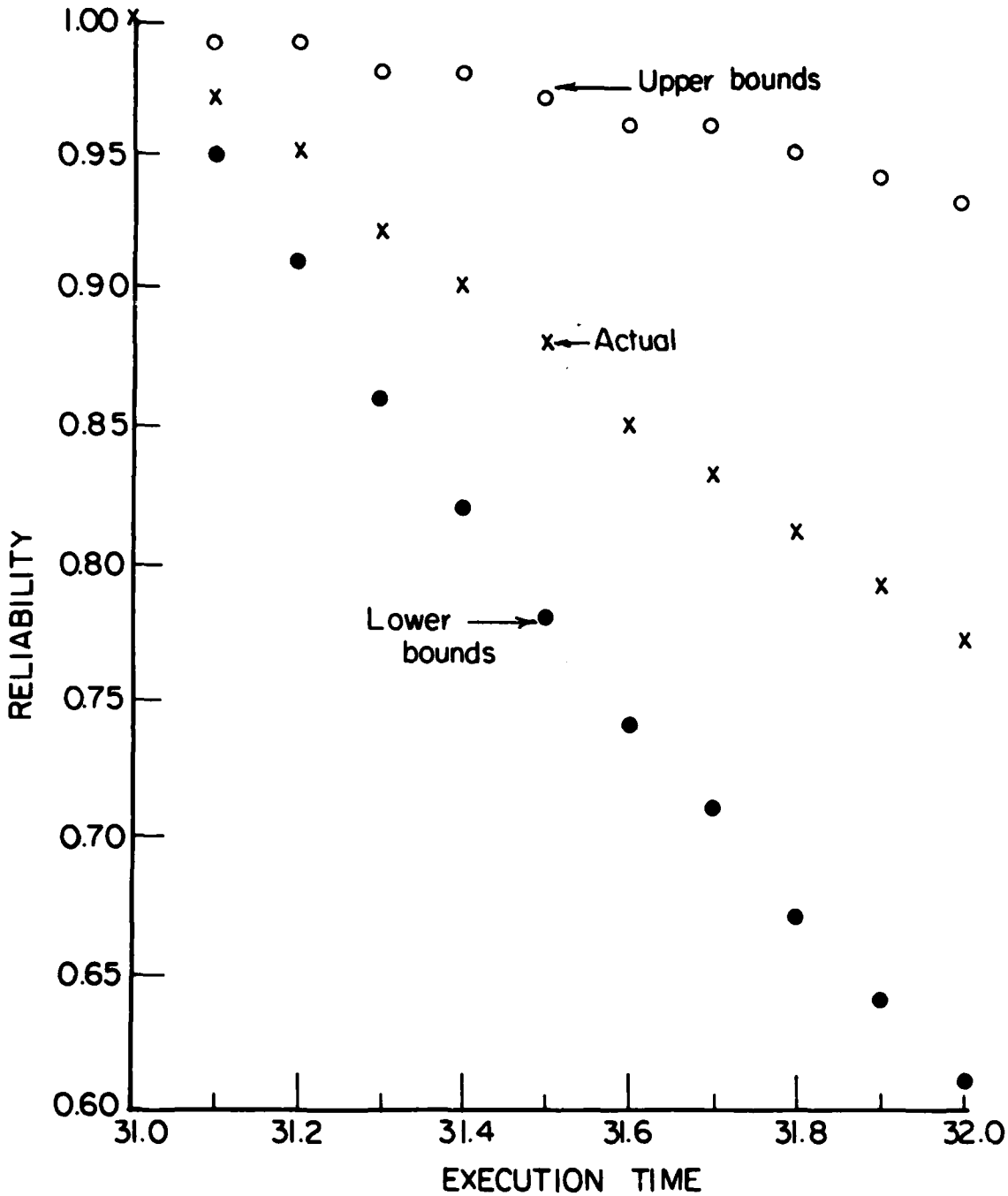


FIG. 2.18 RELIABILITY AND 90% CONFIDENCE BOUNDS - SYS 2.

2.10 ANALYSES OF VARIOUS TYPES OF ERRORS FROM A REAL-TIME CONTROL SYSTEM

In this section, we study the failure data from a real-time control system for a land-based radar system developed by the Raytheon Company [WIL77]. It was developed in a modular fashion (a total of 109 modules) and nearly all modules were written in JOVIAL/J3. (JOVIAL/J3 is the standard programming language for Air Force Command and Control Applications.) The rest of the modules, chiefly the Executive program, were written in Assembly language. The whole system has a total of 86,780 lines and 49,900 Assembly lines of code. The software system runs in JOVIAL, Raytheon's multiprocessor computer which consists of two identical processors (one utilized as a CPU and the other as an I/O control unit), and 81,920 words of 24-bit core memory. The software operates under the control of a highly centralized modular Executive program which supervises all real-time activity on both the CPU and IOCU. The software system features a common data base whose overall layout is defined by means of a COMPOOL. During compile time, the JOVIAL compiler creates the necessary linkages for operational programs to gain access to the data base.

Testing of the software system proceeded in three phases: unit testing of individual program modules, including the Executive program; integration (build) testing; and operational testing of the system in the field. Unit testing was carried out on a Digital System simulator rather than on the live computer in order to take advantage of the simulator's extensive debugging tools. On the other hand, integration testing, whose chief purpose was to check out control and data interfaces among program modules, was done on a real machine. Finally, operational testing was performed on a series of increasingly demanding missions designed to exercise the system and evaluate its response under various loads and physical environments. Operational missions were first rehearsed in conjunction with a mission simulator, then performed with a full hardware complement under actual field conditions.

2.10.1 Error Data

Integration testing was responsible for the largest number of Software Problem Reports (SPR's). The SPR forms were filled by anyone (systems analyst, programmer, or user of the software). SPR's were generated as soon

as an error (problem) was identified and were not delayed until a solution was devised and tested. The error data set used in validating the NHPP model was derived from the SPR's only during the acceptance and operational testing over a 22 month period during 1974-1976. The data for the entire 38 month period will be analyzed in Section 3.

The error data was categorized according to the seriousness of the error as well as according to the type of error as follows.

Seriousness of Error

(1) Critical - if the error is impeding the project development;

(2) Low - if it is not really necessary for a correction to be made for the current develcpment to proceed;

(3) Improvement - if it is a suggestion for improvement but not necessary for satisfactory operation;

(4) Medium - of medium severity.

The number of errors for this classification is given in Table 2.9.

TABLE 2.9

ERROR DATA BASED ON LEVELS OF SERIOUSNESS

Seriousness	Description	Actual number of errors detected during software testing (22 months)	$\hat{a}$	$\hat{b}$
Critical (1)	It is impeding project development	56	73	0.067
Low (2)	It is not really necessary for a correction to be made for the current development to proceed	57	58	0.209
Improvement (3)	It is a suggestion for improvement but not necessary for satisfactory operation	139	142	0.176
Medium (4)	Medium severity	747	785	0.138
TOTAL		999	1046	0.141

Type of Error

Category	Description of Error
A	Computational
B	Logic
D	Data Handling
L	User Requested Changes
M	Preset Data Base
P	Recurrent
E	Others, such as operating system/ support software error, routine/ system interface errors, user interface errors, unidentified errors, etc.

The total number of errors for these categories are given in Table 2.10.

Using the model and estimation technique of Sections 2.2 and 2.5, respectively, the estimated values of a and b were obtained and are also shown in Tables 2.9 and 2.10 for each category of errors. Thus, for critical errors the estimates are $\hat{a} = 73$ and $\hat{b} = 0.067$ and the fitted NHPP is

$$P\{N(t)=y\} = \frac{\{73(1-e^{-.067t})\}^{y} \cdot e^{-73(1-e^{-.067t})}}{y!}, \quad y = 0,1,\ldots .$$

TABLE 2.10

ERROR DATA BASED ON TYPES OF ERRORS

Category	Types of Errors	Actual number of errors detected during software testing (22 months)	$\hat{a}$	$\hat{b}$
Computational (A)	· Errors in computing entry #, indices, and flag settings. · Incorrect/inaccurate equation used, et al.	45	60	0.064
Logic (B)	· Missing logic or condition test · Incorrect logic et al.	178	186	0.141
Data Handling (D)	· Data initialization/ setting error · Data location error et al.	165	169	0.167
User Requested Changes (L)	· Data related errors · Interface design poor et al.	394	421	0.125
Preset Data Base (M)	· Nominal constants · Error messages	80	83	0.152
Recurrent (P)	· Redetected error · Duplicates of previous error et al.	22	22	0.230
Others (E)	· Unidentified errors · I/O errors, et al.	115	117	0.182
TOTAL		999	1046	0.141

Since the observed number of critical errors in 22 months is 56, this model indicates that 73-56 = 17 critical errors are still remaining in the system.

Plots of the actual and fitted values of the number of errors for each category are given in Figures 2.19 to 2.22, respectively. Comparing the actual and fitted curves, the NHPP model seems to provide a satisfactory description of these errors.

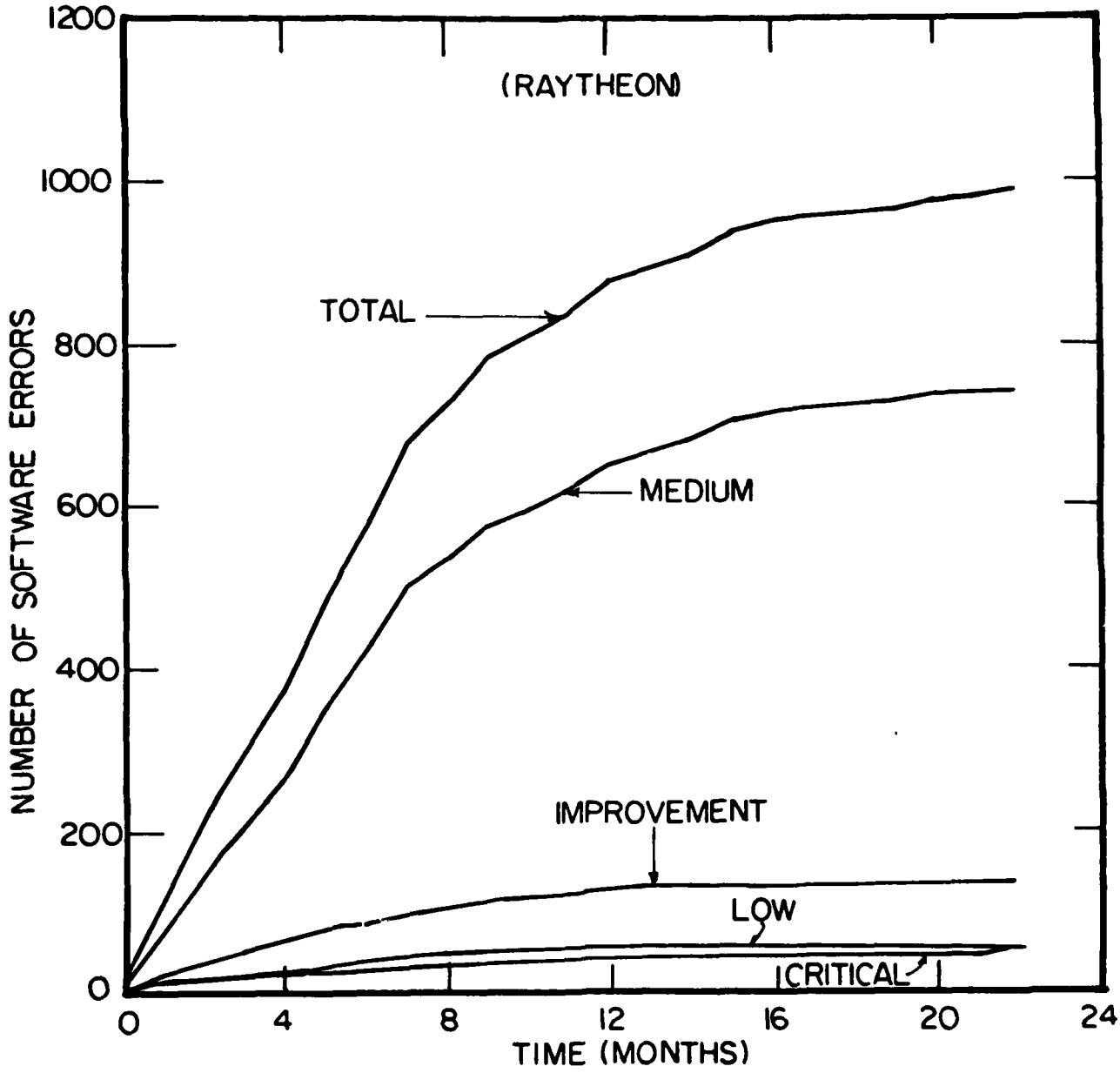


Figure 2.19 Actual software errors with several levels of seriousness during 22-month period of testing

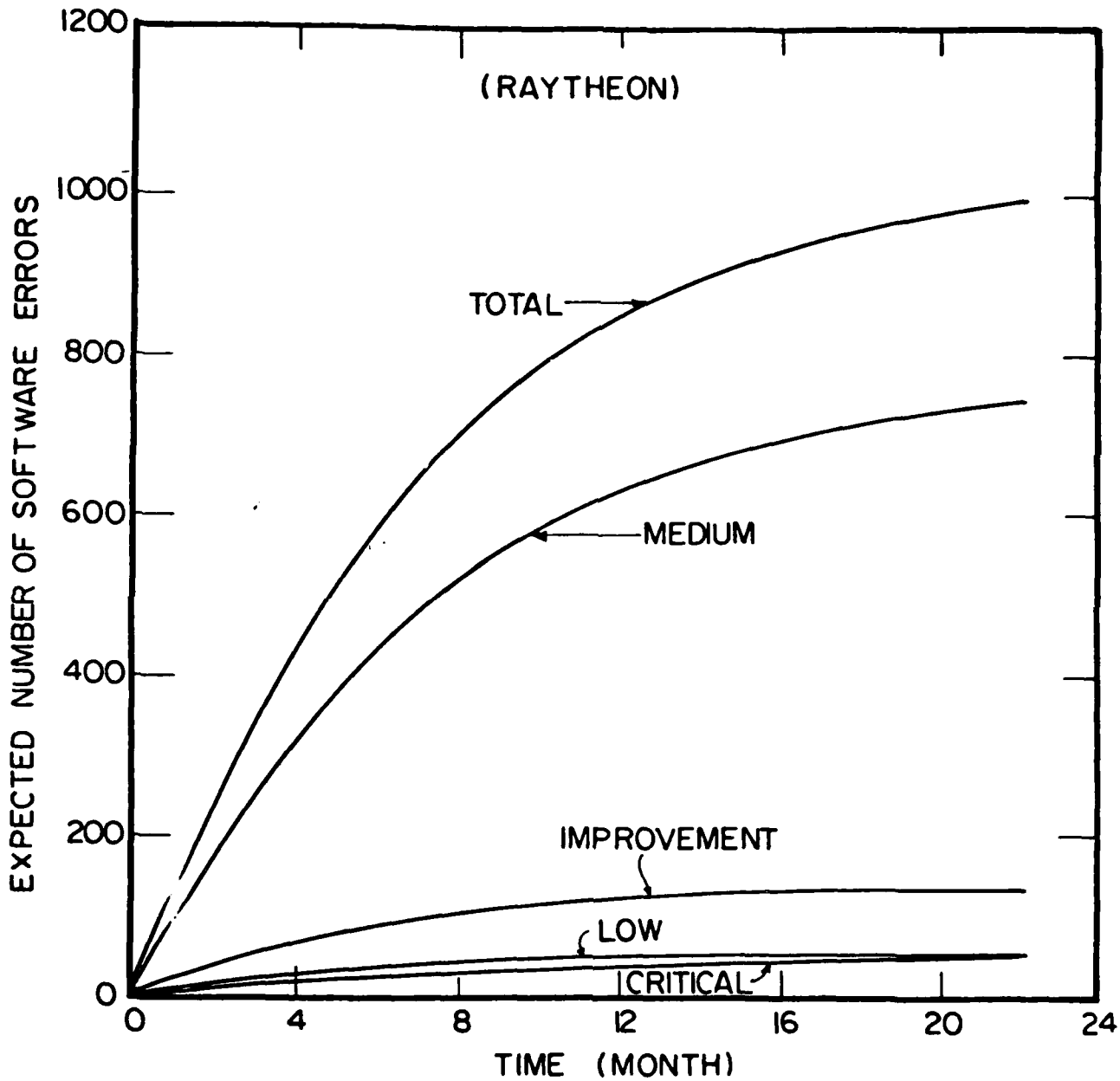


Figure 2.20 The fitted mean value functions for several levels of seriousness

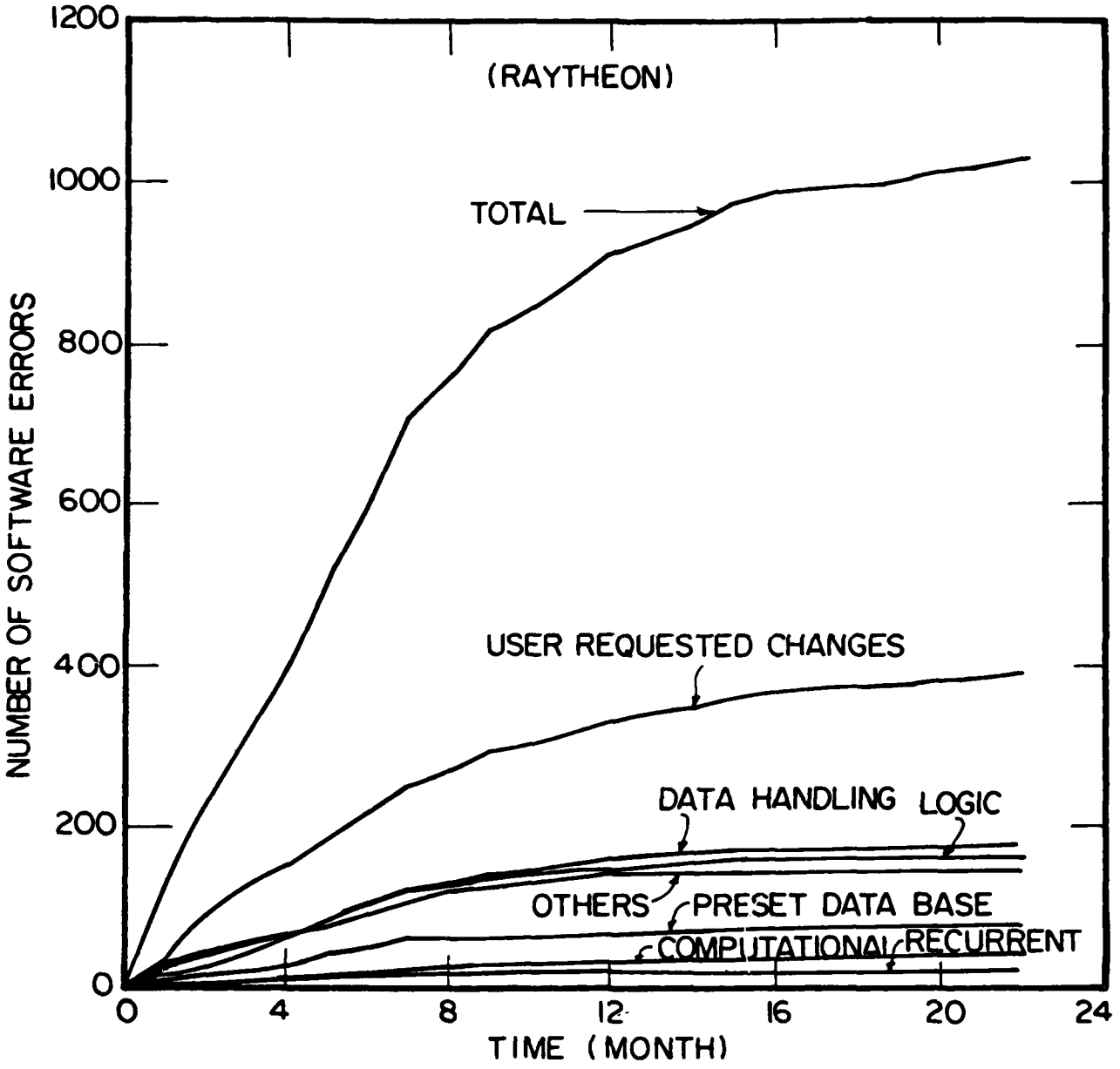


Figure 2.21 Actual software errors of several types during 22-month period of testing

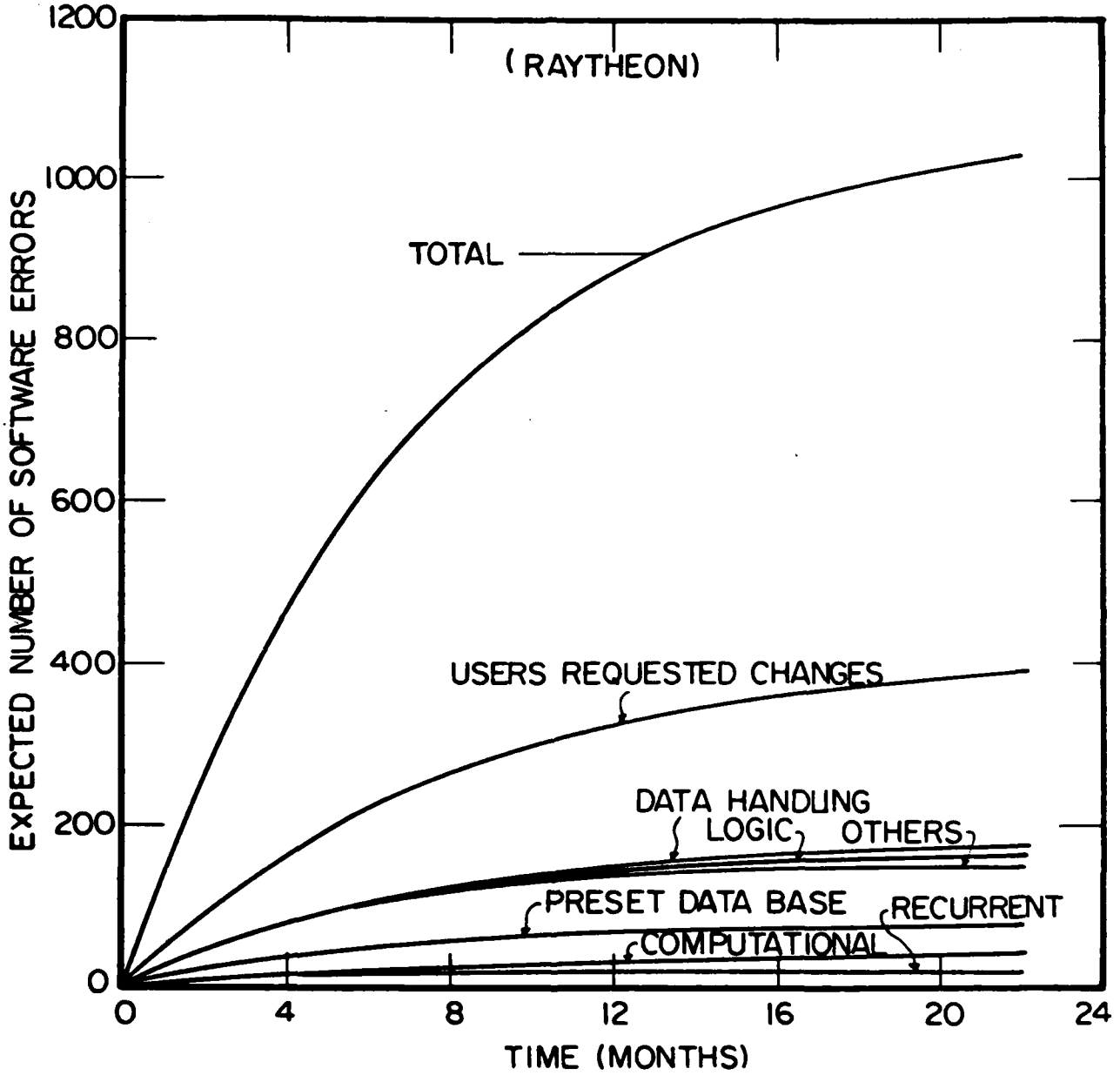


Figure 2.22 The fitted mean value functions for several types of errors

2.11 ANALYSIS OF FAILURE DATA FROM THE APOLLO PROJECT

Now we analyze the failure data from an on-board Apollo space flight software project developed by the Charles Stark Draper Laboratory, Inc. [RYD77] during the years 1967 to 1971. This software, with a size of 83,866 words, runs on the Apollo Guidance Computer (AGC) (designed by MIT/IL) which was used throughout all the Apollo, Skylab, Apollo-Soyuz, and F-8 Phase I programs. The purpose of the AGC was to compute guidance, targeting, navigation, and connrol functions for the Apollo space vehicle for all mission phases.

This software was developed by a group of guidance, navigation, and control engineers, programmers, and test engineers. The coding was done both in the assembly language of the AGC and in the interpretive language (INTERPRETER) developed for the project.

Testing and verification at the laboratory were performed using various facilities, including engineering simulation in the host computer, full scale digital simulation on the host computer, and a hybrid laboratory and system test laboratory that provided real-time execution. Several levels of testing were performed:

Level 1 tests were high order language programs run on the host computer to test algorithms.

Level 2 was the AGC counterpart of these programs.

Level 3 was intended to verify the operation of a complete program or routine including crew interface and realistic physical environment models.

Level 4 testing was intended to verify mission phases, e.g., ascent, redezvous.

Level 5 repeated the level 4 tests on the final rope which was released for manufacture.

Level 6 took place after the ropes were released for manufacture and were intended to verify the program using actual mission data and the flight time-line.

The hybrid and system test laboratories were extensively used in parallel with digital simulation for level 3, 4, 5, and 6 tests. Levels 1 and 2 were performed exclusively on the digital or engineering simulators.

Changes to the software (as a result of software errors) were controlled by the following documents:

- Program Change Request (PCR)
- Program Change Notice (PCN)

- Anomaly Report

- Assembly Control Board Request

The error data set which was derived from these documents was categorized according to types and is summarized in Table 2.11.

The estimates of the model parameters for each category and the total were obtained by the method of Section 2.5 and are given in Table 2.11. The likelihood surface (for total errors) is shown in Figure 2.23 and a plot of the contours of this surface in the (a-b) plane is given in Figure 2.24. From these figures, we note that the surface is really well behaved.

Plots of the observed and estimated total number of failures over the 35 month period are shown in Figures 2.25 and 2.26, respectively. Again, a comparison of the two sets of figures indicates that the model provides an excellent fit to the data.

TABLE 2.11

NUMBER OF VARIOUS TYPES OF ERRORS

AND ESTIMATES OF a AND b

Category	Total No. of Errors	$\hat{a}$	$\hat{b}$
Computation	171	173.95	0.1165
Logic	743	799.82	0.0765
Data Handling	265	305.25	0.0579
User Requested Changes	419	586.43	0.0358
Present Data Base	157	179.79	0.0590
Recurrent	58	60.52	0.0908
Hardware	871	1186.86	0.0378
Others	1650	1754.01	0.0807
Total	4337	4840.30	0.0647

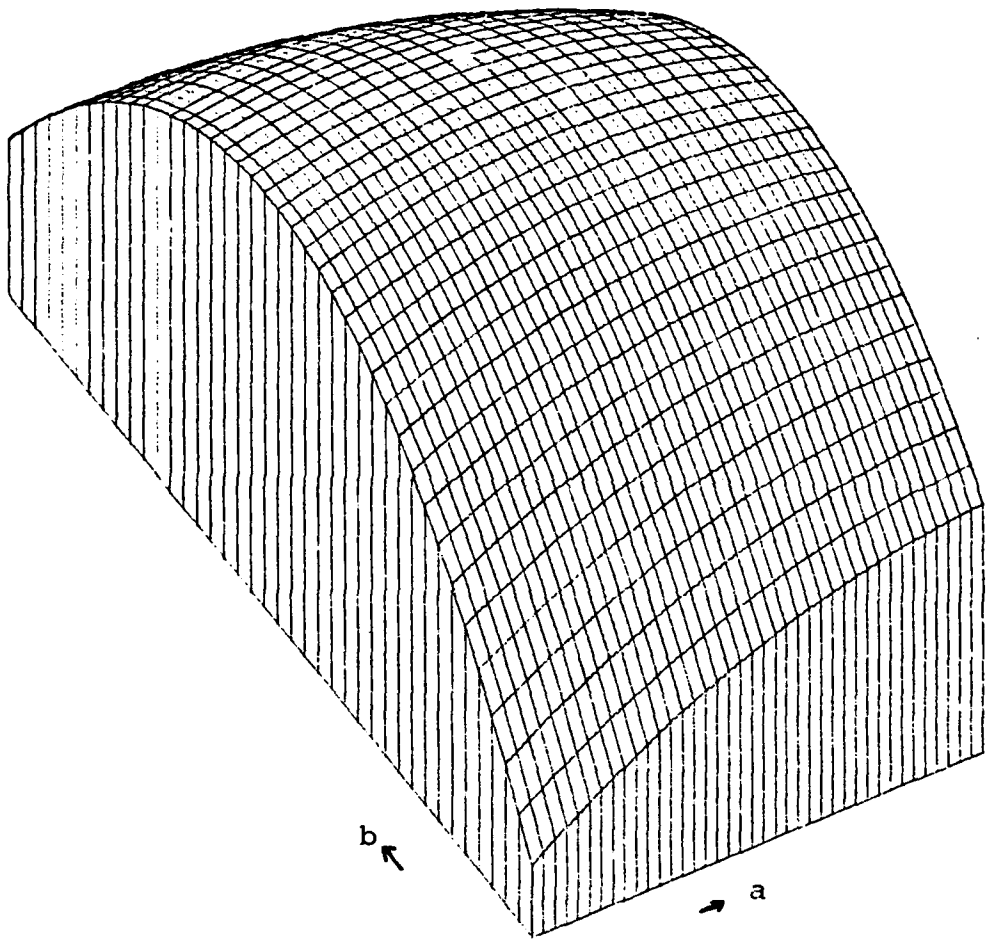


Figure 2.23 Likelihood Surface (Total Errors)

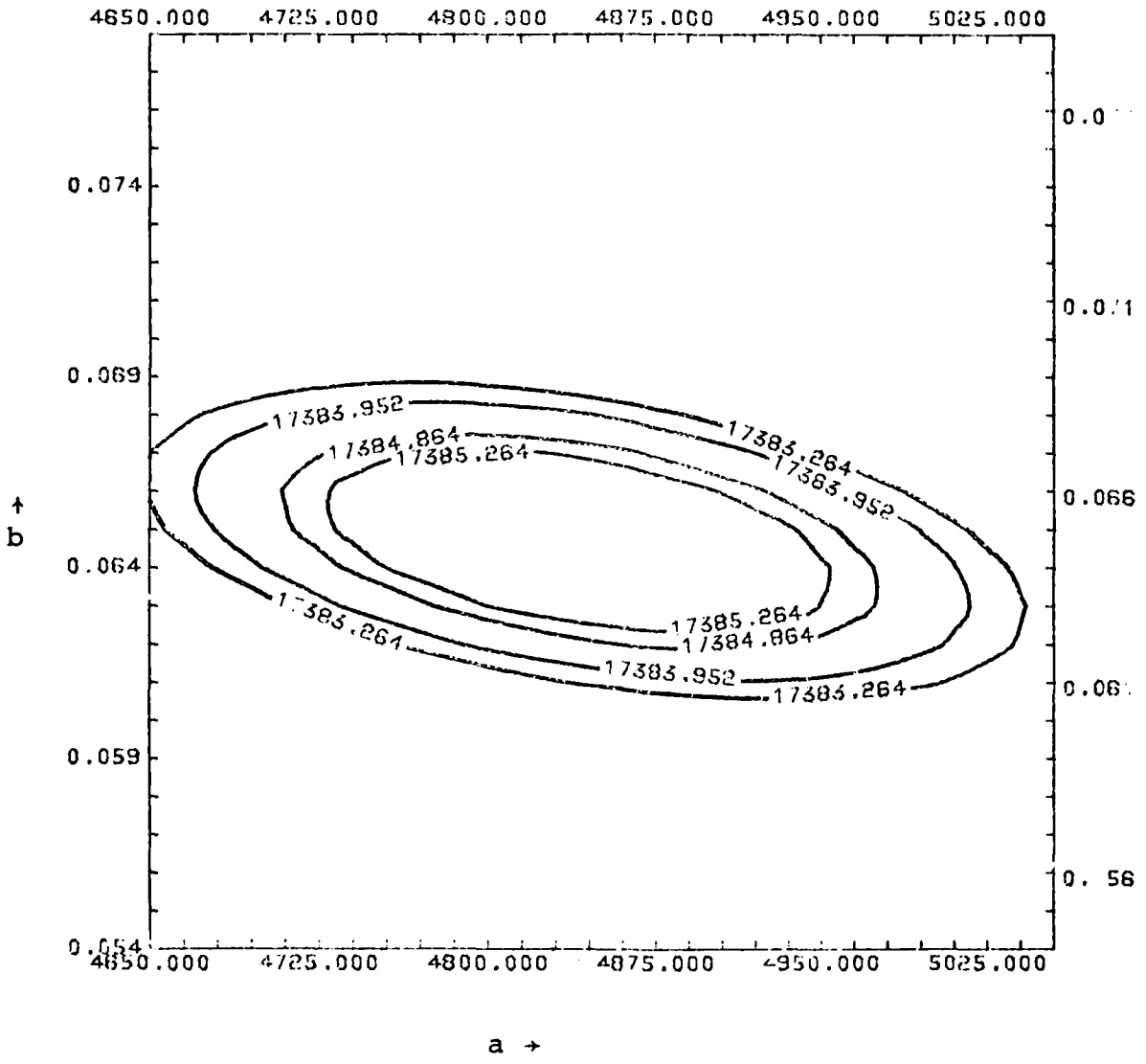


Figure 2.24 Contours of the Likelihood Surface in the (a - b) Plane

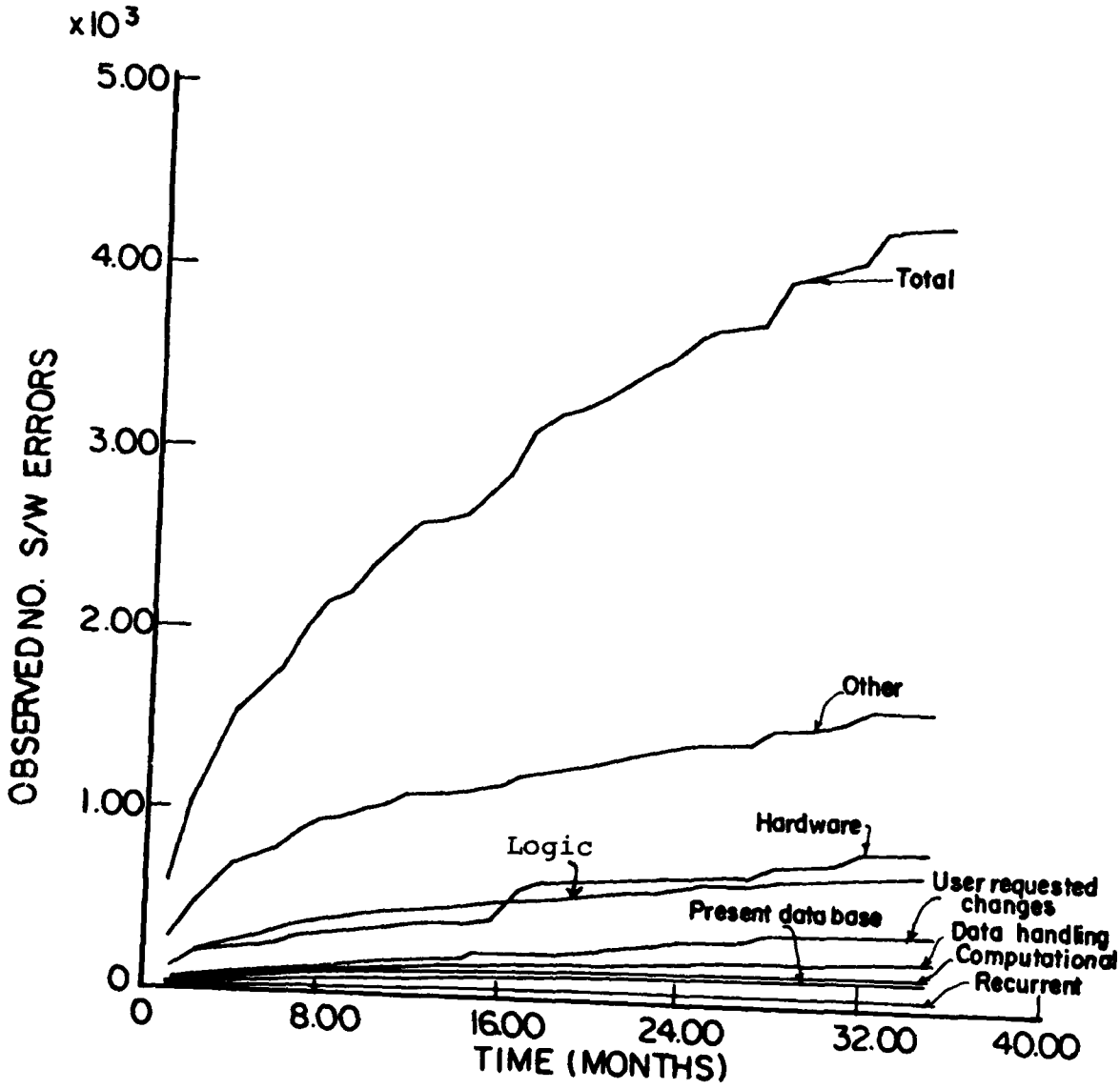


FIGURE 2.25. OBSERVED NUMBER OF SOFTWARE ERRORS (APOLLO).

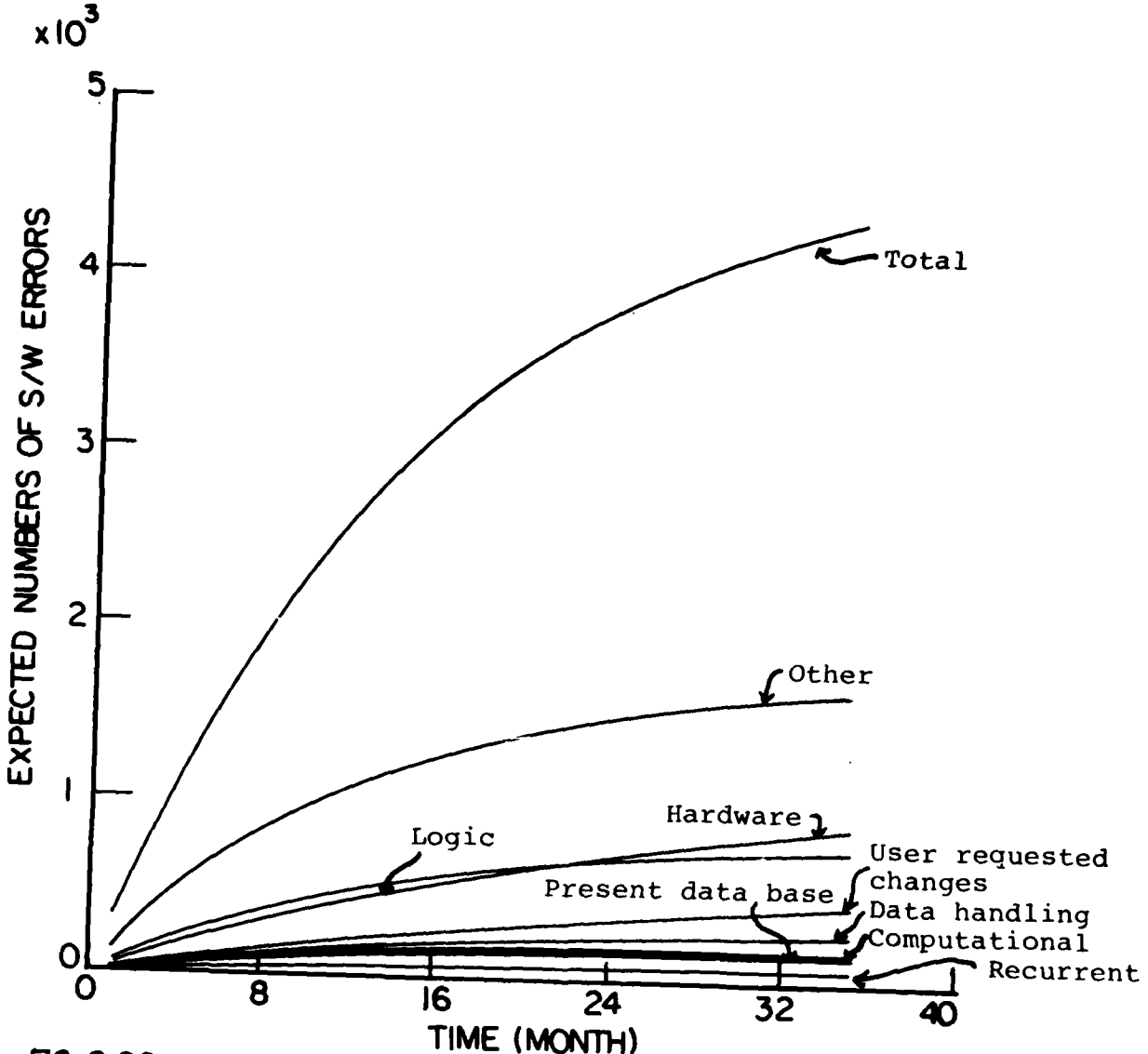


FIG. 2.26 EXPECTED NUMBER OF SOFTWARE ERRORS (APOLLO)

2.12 ANALYSIS OF DATA FROM A LARGE AVIONICS REAL-TIME SYSTEM

The software from which this error (failure) data is taken is a large avionics real-time system for DOD developed by the Boeing Aerospace Company [FRI77]. It consists of 40,640 lines of JOVIAL/J3B instructions and 84,065 assembly language instructions. This system was not developed in modular fashion.

The whole system consists of a controls and displays subsystem, a hardware test monitor, two system functions, and an executive system which schedules the former functions. The software consists of 5 major functional areas in the operational software and two functional areas in the simulation software. The software was designed so that, if one Avionic Control Unit breaks down, the system can still provide the basic functional capabilities. The simulator, which runs on two separate computers, allows testing to take place in the laboratory.

Testing of this software began with Module Verification Testing (MVT) performed by each modules developer. No Software Problem Reports (SPR's) were issued during MVT because, as far as configuration management is concerned, the software was not released yet. Upon comple-

tion of MVT, the developers released the modules for formal testing. Formal testing began with Inter-Module Compatibility Testing (IMCT) where the software was checked against its functional requirements as a total unit. Upon completion of IMCT, the software development group gave the software system to an independent system test group for System Validation Testing (SVT) where acceptance testing for quality control purposes was performed. When an error was discovered during testing, the usual procedure was to patch the program. Software errors were documented on software problem reports (SPR) while requirement errors were reported on Design Change Requests. The data set obtained for this analysis was from the two formal test phases and was both from the operational and simulation software for the first two versions (called blocks) of the software system.

Time to fix an error was calculated based on the number of days an SPR was open and an assumed 8 hour/day of equipment use to fix. This 8 hours was divided up among the errors open on any one day, and this fractional time was summed up over the days the SPR was open, to give the final total time spent fixing an error.

The error data set for the analysis was collected during the period October 1974 to August 1975 on a monthly basis. The errors were categorized into the following groups:

(1) Critical

(2) Low

(3) Improvement

(4) Medium

(5) Other

The total number of errors for each severity level over an eleven month period and the corresponding estimated values of a and b are given in Table 2.12. Plots of the observed and fitted number of errors are shown in Figures 2.27 and 2.28, respectively. Again, the model appears to provide a very good fit to the failure data.

TABLE 2.12

NUMBER OF ERRORS BY SEVERITY

	Severity	Total Errors	$\hat{a}$	$\hat{b}$
1.	Critical	28	33.40	0.1657
2.	Low	51	63.20	0.1495
3.	Improvement	211	260.02	0.1517
4.	Medium	357	501.99	0.1129
5.	Other	780	1031.43	0.1283
	TOTAL	1427	1880.71	0.1293

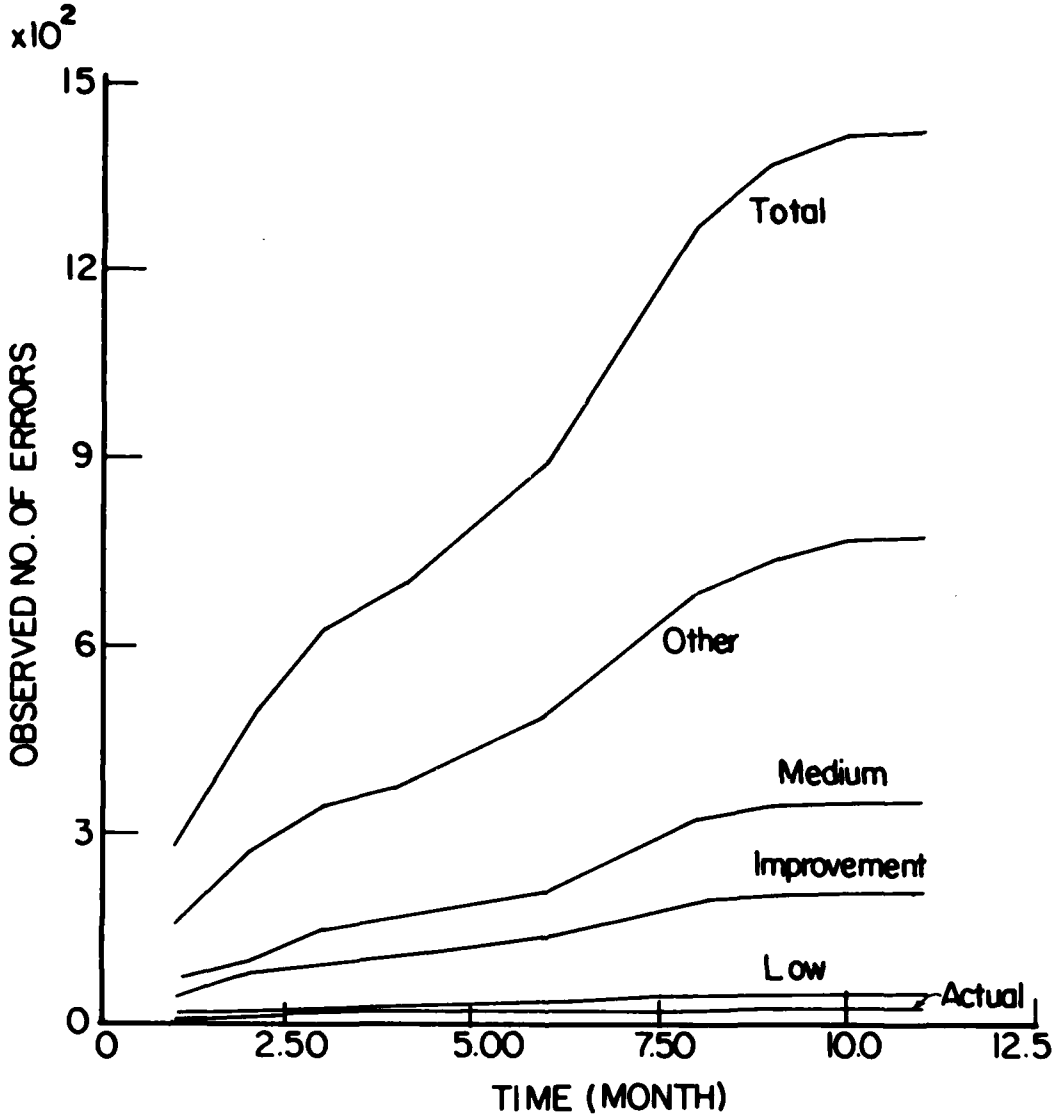


FIG. 2.27 OBSERVED NO. OF ERRORS BY SEVERITY (BOEING)

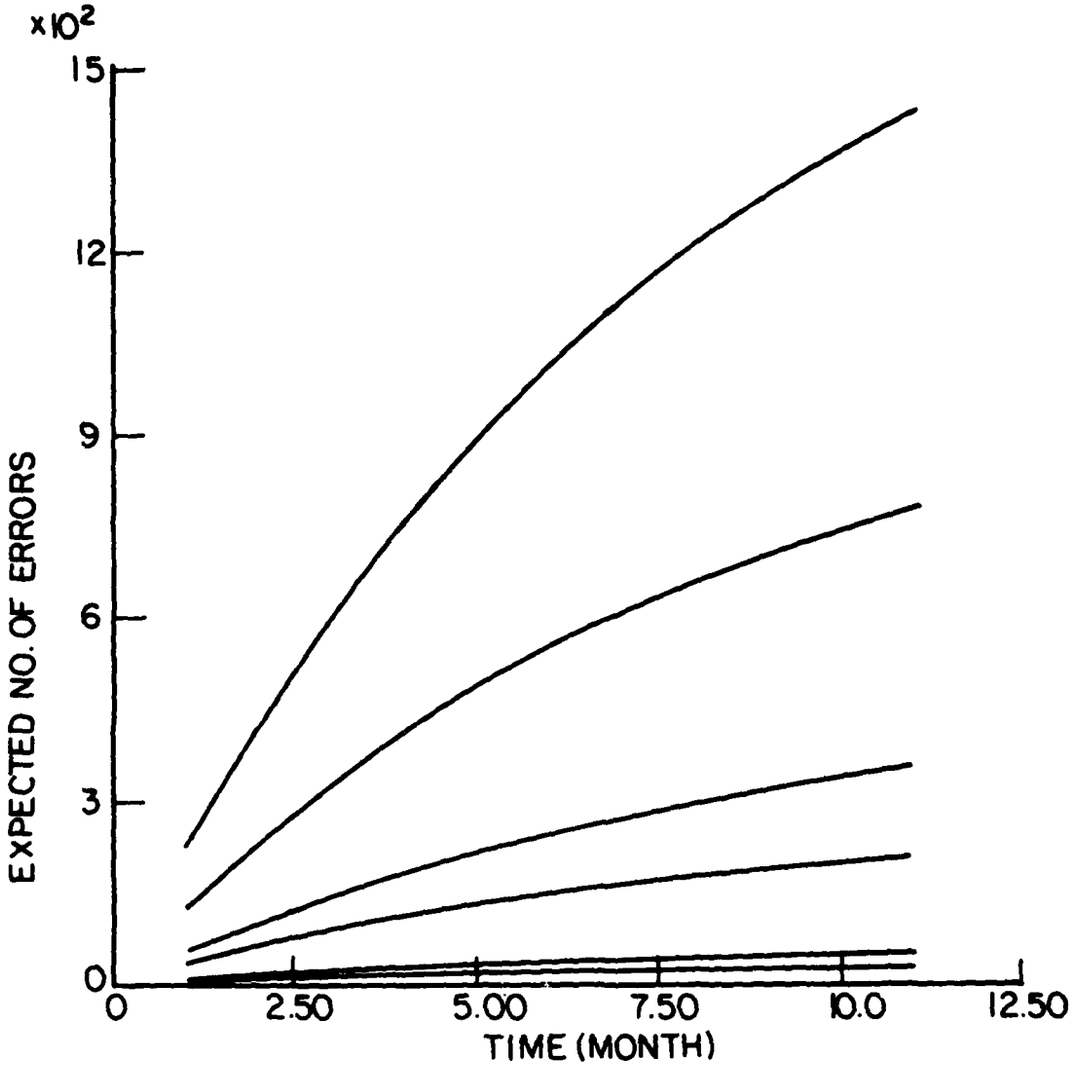


FIG. 2.28 EXPECTED NUMBER OF ERRORS BY SEVERITY.

SECTION 3

SOFTWARE FAULT OCCURRENCE PROCESS WITH INCREASING/DECREASING ERROR DETECTION RATE

3.1 INTRODUCTION

As discussed earlier, many stochastic models have been developed during the past ten years to describe the fault occurrence phenomenon in a large scale software system. Most of these models are based on the assumption that the time between system failures follows an exponential distribution with a parameter that depends either on the number of faults remaining in the system or on the elapsed execution or calendar time. A summary of these models and a comparative list of the features of some of these models was given in section 1.5.

All the models that have been proposed to date make an important assumption about the monotonicity of the software failure rate. In particular, it has been assumed that the software system experiences an improvement with time. In other words, the existing models assume that the software has a decreasing failure rate (DFR). However, in practice, it has been observed that many software systems first experience an increasing failure rate (during

the initial phases of integration) and then follow a decreasing failure rate.

In this section we develop a new model which incorporates this dynamic behavior of the software systems. The basic model is presented in section 3.2 and various software effectiveness measures are developed in section 3.3. Software reliability and related results are given in section 3.4. Methods for estimating the parameters of the model from software failure data are described in section 3.5. Analyses of software failure data from a large scale system and the Naval Tactical Data System are presented in sections 3.6 and 3.7, respectively.

Data sets from numerous other systems were analyzed to assess the applicability of this model. Also, goodness-of-fit tests were conducted following the method discussed in section 2.6. Details of these analyses and tests are not reported here for the sake of brevity. In all of the cases studied the model reported here was found to provide an excellent fit to the observed failure history.

3.2 MODEL DEVELOPMENT

In order to develop an appropriate model, we study the stochastic behavior of the fault detection phenomenon by focusing our attention on the number of faults detected by some arbitrary time t. Let N(t) denote the number of faults detected by time t and let m(t) be the expected value of N(t), i.e.,

$$m(t) = E[N(t)] \ . \qquad (3.1)$$

The above function m(t) is called the mean value function of the N(t) process. It should be pointed out that here time t can be calendar time, execution time, or any other suitable and consistent measure of time. In practice, however, we have found calendar time and CPU time as the commonly used measures.

3.2.1 Assumptions

We now consider the behavior of the software fault detection process as described by N(t).

(i) There will be no faults detected at the beginning of the fault detection process, i.e., we

have $N(0) = 0$. Also, this implies

$$m(0) = 0. \tag{3.2}$$

(ii) It is quite obvious that the software system must contain a finite number of faults. In other words, if testing were to be continued indefinitely, the number of faults to be detected will be finite, so that the expected number of faults to be eventually found will be $m(\infty)$. Let

$$m(\infty) = a < \infty. \tag{3.3}$$

(iii) The faults to be detected are such that each one effects the failure occurrence phenomenon independently of others, but the rate at which each fault causes the system to fail depends on elapsed time. This can be expressed by taking the hazard rate $z(t)$ of each fault to be

$$z(t) = bct^{c-1}. \tag{3.4}$$

Note that the shape of this function will depend on the values of the parameters b and c.

3.2.2 Expression for m(t)

Based on the above description of the fault detection process, we now develop an expression for m(t). In terms of m(t), the hazard rate at time t is defined as

$$z(t) = \frac{m(t+\Delta t) - m(t)}{\Delta t\{a-m(t)\}} .$$

Substituting for z(t) from Equation (3.4), we get

$$\frac{m(t+\Delta t) - m(t)}{\Delta t\{a-m(t)\}} = bct^{c-1} . \qquad (3.5)$$

By letting $\Delta t \to 0$ in the above equation, we get a first-order linear differential equation

$$m'(t) + bct^{c-1}m(t) = abct^{c-1} . \qquad (3.6)$$

To solve the above equation for m(t), we need to use the following results.

Lemma. If P(t) and Q(t) are two continuous functions of t, then the general solution of an equation of the form

$$y' + P(t)y = Q(t) \tag{3.7}$$

is

$$y = \frac{1}{h(t)} \int Q(t)h(t)dt, \tag{3.8}$$

where

$$h(t) = e^{\int P(t)dt} . \tag{3.9}$$

Proposition. Under the boundary condition $m(0) = 0$, the solution of equation (3.6) is given by

$$m(t) = a(1-e^{-bt^c}) . \tag{3.10}$$

Proof. Let the functions P(t) and Q(t) in the above Lemma be

$$P(t) = bct^{c-1}$$

and

$$Q(t) = abct^{c-1} .$$

Then $h(t)$ is obtained from (3.9) as

$$h(t) = e^{\int P(t)dt}$$

or

$$h(t) = e^{bt^c} \tag{3.11}$$

and

$$\int Q(t)h(t)dt = \int abct^{c-1}e^{bt^c}dt$$

or

$$\int Q(t)h(t)dt = ae^{bt^c} + k , \tag{3.12}$$

where k is a constant to be determined by the boundary condition $m(0) = 0$. Finally, we get the solution of

(3.6) by substituting (3.11) and (3.12) into (3.8), i.e.,

$$m(t) = e^{-bt^c}(ae^{bt^c} + k)$$

or

$$m(t) = a + ke^{-bt^c} . \qquad (3.13)$$

Since $m(0) = 0$, we have

$$m(0) = a + k = 0$$

or

$$k = -a .$$

Substituting $k = -a$ in (3.13), we get the result of Equation (3.10).

3.2.3 Fault Detection Rate

Fault detection rate is the number of faults per unit time. Let $\lambda(t)$ denote the software fault detec-

tion rate so that, for a small time interval Δt, $\lambda(t)\Delta t$ represents the number of software faults detected during $(t, t+\Delta t)$. Now $m(t)$ is the expected number of faults detected by t and

$$\lambda(t) = m'(t) \ . \tag{3.14}$$

From Equations (3.10) and (3.14), we get

$$\lambda(t) = abt^{c}r^{-bt^{c}}ct^{c-1}$$

or

$$\lambda(t) = \alpha t^{\gamma-1} \cdot e^{-\beta t^{\gamma}} \tag{3.15}$$

where

$$\begin{aligned} \alpha &= abc \\ \beta &= b \\ \gamma &= c \end{aligned} \tag{3.16}$$

In order to see the shape of the fault detection rate $\lambda(t)$, we differentiate Equation (3.15) with respect to t and equate the result to zero and get

$$t^{\gamma} = \frac{\gamma-1}{\beta_{\gamma}} \ . \qquad (3.17)$$

We see that, for $r > 1$, $\lambda(t)$ is a unimodal function with

$$(0) = \lambda(\omega) = 0 \ ,$$

and its maximum value occurs at $t = t_m$ where

$$t_m = (\frac{\gamma-1}{\beta\gamma})^{1/\gamma}. \qquad (3.18)$$

The maximum value of $\lambda(t)$ is

$$\max\lambda(t) \equiv \lambda(t_m) = \alpha(\frac{\gamma-1}{\beta\gamma})^{(\gamma-1)/\gamma} \cdot e^{-(\gamma-1)/\gamma}. \qquad (3.19)$$

In other words, the error detection rate of software or, equivalently, the software failure rate, increases during the period $(0,t_m)$, achieves its maximum value $\lambda(t_m)$ at $t = t_m$, and then decreases for $t > t_m$ eventually becoming zero at $t = \infty$. Note that if $0 < \gamma < 1$, then the software failure rate is monotonically decreasing. From the above discussion, we see that the software fault detection rate $\lambda(t)$ is increasing/decreasing if $r > 1$, and monotonically decreasing if $0 < \gamma < 1$.

The extreme values are $\lambda(0) = \alpha$ for $\gamma = 1$ and $\lambda(0) = \infty$ for $0 < \gamma < 1$.

3.2.4 Failure Counting Process

Now we assume that the failure counting process N(t) has the following characteristics:

(i) N(t) has independent increments, i.e., $\{N(t_2) - N(t_1)\}$ is independent of $\{N(t_3) - N(t_2)\}$ for some $t_1 < t_2 < t_3$.

(ii) The probabilities associated with the N(t) process are as follows:

$$N(t+\Delta t) - N(t) = \begin{cases} 0 \text{ with probability } 1-\lambda(t)\Delta t + 0(\Delta t) \\ 1 \text{ with probability } \lambda(t)\Delta t + 0(\Delta t) \\ 2 \text{ with probability } 0(\Delta t) \end{cases} \quad (3.20)$$

It is well known that with the above properties and with $\lambda(t)$ as given in Equation (3.15), the N(t) process is a non-homogeneous Poisson process (NHPP) with a mean value function m(t) given in Equation (3.10). Hence, the distribution of N(t) is given by

$$P\{N(t) = y\} = \frac{\{m(t)\}^y}{y!} \cdot e^{-m(t)} \quad . \qquad (3.21)$$

Under the assumptions discussed above, the stochastic behavior of the software failure phenomenon can be completely described by the model given in Equations (3.10) and (3.21). These equations constitute the basic failure occurrence model discussed in this section.

3.3 SOFTWARE EFFECTIVENESS MEASURES

In this section, we develop expressions for several useful quantitative measures for assessing the software system effectiveness.

3.3.1 Distribution of the Number of Faults Detected or Failures Observed

As indicated above, N(t) is a NHPP with a probability mass function

$$P\{N(t) = y\} = \frac{\{a(1-e^{-bt^c})\}^y}{y!} e^{-a(1-e^{-bt^x})}, \quad y = 0,1,2,... \qquad (3.22)$$

As $t \to \infty$, we have

$$P\{N(\infty) = y\} = \frac{a^y}{y!} e^{-a}, \quad y = 0,1,2,... \qquad (3.23)$$

This last expression tells us that, if the system were to be used for a long time ($t = \infty$), the number of faults detected or failures observed during this time follows a Poisson process with mean 'a'.

3.3.2 Number of Faults Remaining in the System

Let $\bar{N}(t)$ denote the number of faults not detected by time t, i.e., the number of faults remaining in the system. Clearly, this number will be obtained by subtracting N(t) from N(∞), the number of faults to be eventually detected. Note that these quantities are random variables. Thus, we have

$$\bar{N}(t) = N(\infty) - N(t) \tag{3.24}$$

and

$$E[\bar{N}(t)] = E[N(\infty)] - E[N(t)]$$

or

$$E[\bar{N}(t)] = a - a(1-e^{-bt^c})$$

or

$$E[\bar{N}(t)] = ae^{-bt^c} . \tag{3.25}$$

3.3.3 Conditional Distribution of $\bar{N}(t)$

If we have already observed y faults, it is useful to know the distribution of the number of faults yet to be detected. In other words, the conditional distribution of $\bar{N}(t)$, given that $N(t) = y$, is

$$P\{\bar{N}(t) = x | N(t) = y\} = \frac{P\{\bar{N}(t) = x,\ N(t) = y\}}{P\{N(t) = y\}} . \quad (3.26)$$

Now the event $\bar{N}(t) = x$ denotes occurrences over the time interval (t,∞) while the event $N(t) = y$ denotes occurrences over the interval $(0,t)$, i.e., these two events represent non-overlapping time intervals. From a basic property of the NHPP process, such events are independent of each other, so that we have

$$P\{\bar{N}(t) = x | N(t) = y\} = P\{\bar{N}(t) = x\},\ x=0,1,2,... \quad (3.27)$$

or

$$P\{N(\infty) - N(t) = x | N(t) = y\} =$$

$$\frac{\{m(\infty)-m(t)\}^x}{x!} \cdot e^{-\{m(\infty)-m(t)\}} .$$

Or, substituting for $m(\infty)$ and $m(t)$ from Equation (3.10), we get

$$P\{N(\infty) - N(t) = x \mid N(t) = y\} =$$

$$\frac{\{a-a(1-e^{-bt^c})\}^x}{x!} \cdot e^{-\{a-a(1-e^{-bt^c})\}} .$$

This yields

$$P\{\bar{N}(t) = x \mid N(t) = y\} = \frac{\{ae^{-bt^c}\}^x}{x!} \cdot e^{-ae^{-bt^c}} . \quad (3.28)$$

Finally, the expected number of faults to be detected, given $N(t) = y$, is

$$E[\bar{N}(t) \mid N(t) = y] = ae^{-bt^c} . \quad (3.29)$$

3.3.4 Joint Counting Probability

The property of independent increments, along with the equations developed above, provides a complete statistical characterization of the NHPP process so that the joint probability of certain number of faults occurring in given time intervals is obtained as follows. Consider times $t_1, t_2, \ldots, t_n$ such that $0 < t_1 < t_2 < \ldots < t_n$. We have, with $t_0 = 0$, $y_0 = 0$,

$$P\{N(t_1) = y_1,\ N(t_2) = y_2,\ \ldots,\ N(t_n) = y_n\}$$

$$= \prod_{i=1}^{n} P\{N(t_i) - N(t_{i-1}) = y_i - y_{i-1}\} \qquad (3.30)$$

$$= \prod_{i=1}^{n} \frac{\{m(t_i)-m(t_{i-1})\}^{y_i-y_{i-1}}}{(y_i-y_{i-1})!} \cdot e^{-\{m(t_i)-m(t_{i-1})\}}$$

Equation (3.30) will be used for estimating the parameters a, b, and c from given failure data in later sections.

3.4 SOFTWARE RELIABILITY AND DISTRIBUTION OF TIME BETWEEN SOFTWARE FAILURES

The time between failures is a stochastic process whose behavior is governed by many factors such as the usage of the system, system load, degree of purification of software, etc. However, since it is not presently feasible to quantify the effects of these factors individually, we model the process behavior as described above, i.e., by a NHPP process with an increasing/decreasing fault detection rate. At any given point, the time to next failure will depend on the time when the last failure occurred. Suppose that the (k-1)st failure occurred at some time $S_{k-1} = s$. Then the probability that the kth failure will not occur for an additional time $X_k = x$, i.e., the conditional probability for time x, is as follows:

$$P(\text{no failure in } (s,s+x) \mid \text{failure at } s)$$

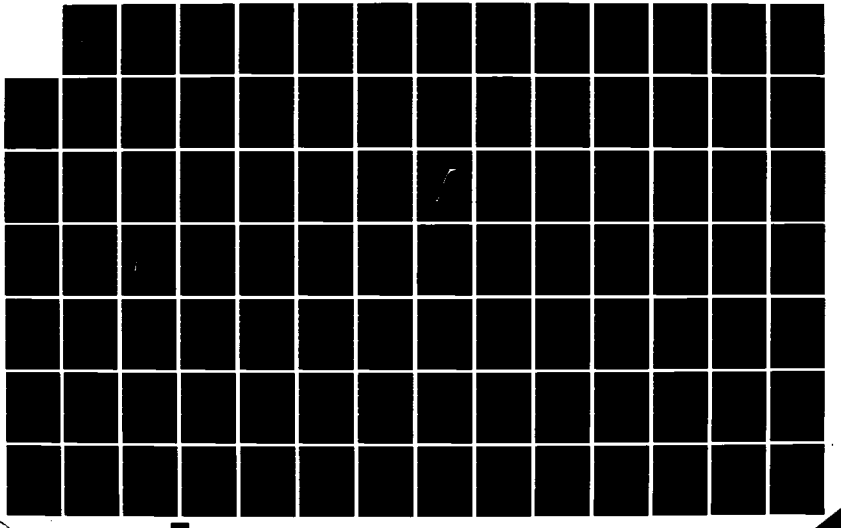
$$= R_{X_k|S_{k-1}}(x|s)$$

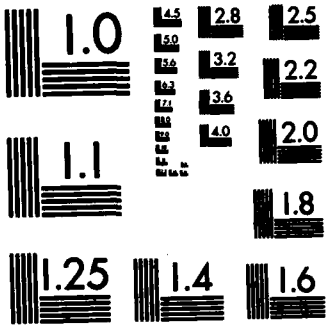
and

$$= R_{X_k|S_{k-1}}(x|s) = e^{-a\{e^{-bs^c} - e^{-b(s+x)^c}\}} \quad . \quad (3.31)$$

AD-A123 421 SOFTWARE RELIABILITY MODELLING AND ESTIMATION TECHNIQUES(U) SYRACUSE UNIV N Y DEPT OF INDUSTRIAL ENGINEERING AND OPERATIONS RESEARCH A L GOEL OCT 82 3/4

UNCLASSIFIED RADC-TR-82-263 F30602-78-C-0351 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Since the conditional cumulative distribution function (cdf) is related to conditional reliability by

$$F_{X_k|S_{k-1}}(x|s) = 1 - R_{X_k|S_{k-1}}(x|s) , \qquad (3.32)$$

we have

$$F_{X_k|S_{k-1}}(x|s) = 1 - e^{-a\{e^{-bs^c} - e^{-b(s+x)^c}\}}. \qquad (3.33)$$

The conditional probability density function (pdf) is obtained from Equation (3.33) by differentiating $F_{X_k|S_{k-1}}(x|s)$ with respect to x and is given by

$$f_{X_k|S_{k-1}}(x|s) = abc(s+x)^{c-1}e^{-b(s+x)^c}e^{-a\{e^{-bs^c} - e^{-b(s+x)^c}\}} \qquad (3.34)$$

Finally, we are also interested in the joint pdf of the cumulative times to failures, i.e., in the joint pdf of $S_1, S_2, \ldots, S_n$. Following the approach given in Section 2, we get

$$f_{S_1,S_2,\ldots,S_n}(s_1,s_2,\ldots,s_n) = e^{-m(s_n)} \prod_{k=1}^{n} \lambda(s_k), \qquad (3.35)$$

where

$$m(s_n) = a(1-e^{-bs_n^c}) , \qquad (3.36)$$

and

$$\lambda(s_k) = \alpha s_k^{\gamma-1} e^{-\beta s_k^{\gamma}} , \qquad (3.37)$$

or

$$\lambda(s_n) = abc \cdot s_n^{c-1} \cdot e^{-bs_n^c} . \qquad (3.38)$$

These results are used for estimating the parameters a, b, and c in later sections.

3.5 ESTIMATION OF MODEL PARAMETERS FROM FAILURE DATA

In this section, we describe methods for estimating the parameters a, b, and c (or, equivalently, α, β, and γ) from available data on software failures. Such data are generally available either as cumulative number of failures in given time intervals, or as times between software failures. The estimation procedure is different for each case and is described below. In this report, we use the method of maximum likelihood for estimation purposes.

3.5.1 Maximum Likelihood Estimation When Data on Cumulative Software Failures are Given

Let y_1 be the number of failures observed during a time interval $(0,t_1)$, y_2 during the interval $(0,t_2)$, and so on. In general, let y_i be the number of failures by time t_i. Then the observed data in this case will consist of apirs (t_i,y_i), $i = 1,2,\ldots,n$. Now the probability of observing (y_i-y_{i-1}) failures during a time interval (t_i-t_{i-1}) is given by (see Equation (3.30)),

$$P\{N(t_i)-N(t_{i-1}) = y_i-y_{i-1}\}$$

$$= \frac{\{m(t_i)-m(t_{i-1})\}^{y_i-y_{i-1}}}{(y_i-y_{i-1})!} e^{\{-m(t_i)-m(t_{i-1})\}}. \qquad (3.39)$$

Since the increment in the number of failures during the non-overlapping time periods $(0,t_1)$, $(t_1,t_2),\ldots,$ $(t_{i-1},t_i),\ldots,(t_{n-1},t_n)$, are independent of each other, the joint probability of the pairs of observations (t_1,y_1), $(t_2,y_2),\ldots,(t_n,y_n)$, $P\{N(t_1) = y_1, N(t_2) = y_2, \ldots,N(t_i) = y_i,\ldots,N(t_n) = y_n\}$, can be written as

$$P\{N(t_1) = y_1\},\ P\{N(t_2-t_1) = y_2-y_1\},\ldots,$$

$$P\{N(t_i-t_{i-1}) = y_i-y_{i-1}\},\ldots,$$

$$P[N(t_n-t_{n-1}) = y_n-y_{n-1}]$$

$$= \prod_{i=1}^{n} P[N(t_i-t_{i-1}) = y_i-y_{i-1}]$$

$$= \prod_{i=1}^{n} \frac{\{m(t_i)-m(t_{i-1})\}^{y_i-y_{i-1}}}{(y_i-y_{i-1})!}\, e^{-\sum_{i=1}^{n}\{m(t_i)-m(t_{i-1})\}},$$

or

$$P\{N(t_1) = y_1, N(t_2) = y_2,\ldots,N(t_n) = y_n\}$$

$$= \prod_{i=1}^{n} \frac{\{m(t_i)-m(t_{i-1})\}^{y_i-y_{i-1}}}{(y_i-y_{i-1})!}\, e^{-m(t_n)} . \qquad (3.40)$$

From this, the likelihood function for parameters a, b, and c, corresponding to the observations (t_i,y_i), $i = 1,2,\ldots,n$, is obtained as

$$L(a,b,c|(t_1,y_1),(t_2,y_2),\ldots,(t_n,y_n))$$

$$= \prod_{i=1}^{n} \frac{\{m(t_i)-m(t_{i-1})\}^{y_i-y_{i-1}}}{(y_i-y_{i-1})!} e^{-m(t_n)}. \quad (3.41)$$

Taking the natural logarithm on both sides of Equation (3.41), the log likelihood is obtained as

$$\ell(a,b,c|(t_i,y_i),i=1,2,\ldots,n) \equiv \ell n\ L(a,b,c|(t_i,y_i), \quad i = 1,2,\ldots,n)$$

$$= \sum_{i=1}^{n}(y_i-y_{i-1})\ell n\{m(t_i)-m(t_{i-1})\} - m(t_n) - \sum_{i=1}^{n} \ell n(y_i-y_{i-1})! \quad (3.42)$$

On substituting for $m(t_{i-1})$, $m(t_i)$, and $m(t_n)$ from Equation (3.10), and simplifying, the log-likelihood function becomes

$$\ell(a,b,c|(t_i,y_i),\ i = 1,2,\ldots,n)$$

$$= \sum_{i=1}^{n} (y_i - y_{i-1}) \ell n\{a(e^{-bt_{i-1}^c} - e^{-bt_i^c})\}$$

$$- a(1-e^{-bt_h^c}) - \sum_{i=1}^{n} \ell n(y_i - y_{i-1})! \qquad (3.43)$$

It is well known that the maximum likelihood estimates (mle's) $\hat{a}$, $\hat{b}$, and $\hat{c}$, are those values of a, b, and c, respectively, that maximize the likelihood function given in equation (3.42), or equivalently, are those values that maximize the log likelihood function of Equation (3.43). Thus, $\hat{a}$, $\hat{b}$, and $\hat{c}$ are those values that simultaneously satisfy the following equations:

$$\frac{\partial \ell}{\partial a} = 0 \ . \qquad (3.44)$$

$$\frac{\partial \ell}{\partial b} = 0 \ . \qquad (3.45)$$

$$\frac{\partial \ell}{\partial c} = 0 \ . \qquad (3.46)$$

On taking the derivatives of Equation (3.43) with respect to a, b, and c, and substituting in Equations (3.44), (3.45), and (3.46), respectively, we get the

following non-linear simultaneous equations:

$$y_n = a(1 - e^{-bt_n^c}) , \tag{3.47}$$

$$at_n^c e^{-bt_n^c} = \sum_{i=1}^{n} \frac{(y_i - y_{i-1})(t_i^c e^{-bt_i^c} - t_{i-1}^c e^{-bt_{i-1}^c})}{(e^{-bt_{i-1}^c} - e^{-bt_i^c})}, \tag{3.48}$$

and

$$at_n^c (\ell n\ t_n) e^{-bt_n^c} = \sum_{i=1}^{n} \frac{(y_i - y_{i-1})\{t_i^c (\ell n t_i) e^{-bt_i^c} - t_{i-1}^c (\ell n t_{i-1}) e^{-bt_{i-1}^c}\}}{(e^{-bt_{i-1}^c} - e^{-bt_i^c})} . \tag{3.49}$$

The set of simultaneous equations (3.47), (3.48), and (3.49) can be solved numerically for a, b, and c. The solution will be the required maximum likelihood estimates $\hat{a}$, $\hat{b}$, and $\hat{c}$ of a, b, and c, respectively.

3.5.1.1 Variance-Covariance of $\hat{a}$, $\hat{b}$, and $\hat{c}$

Once the estimates of a, b, and c have been obtained from the data, the performance measures, as derived in sections 3.3 and 3.4, can be easily computed by substituting $\hat{a}$, $\hat{b}$, and $\hat{c}$ for a, b, and c, respectively. In order to obtain confidence bounds on the performance measures, we need to know the distribution of the estimates $\hat{a}$, $\hat{b}$, and $\hat{c}$. For a reasonably large sample size n, say $n \geq 20$, the maximum likelihood estimators generally follow a normal distribution. Thus the vector $(\hat{a}\ \hat{b}\ \hat{c})'$ will have a trivariate normal distribution (TVN) with $(a\ b\ c)'$ as the vector of means and Σ_{cov} as the variance-covariance matrix. In other words, for large n,

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \sim \text{TVN} \left(\begin{pmatrix} a \\ b \\ c \end{pmatrix}, \Sigma_{cov} \right). \qquad (3.50)$$

The variance-covariance matrix Σ_{cov} represents

$$\Sigma_{cov} = \begin{pmatrix} \text{Var}(a) & \text{Cov}(a,b) & \text{Cov}(a,c) \\ \text{Cov}(b,a) & \text{V}(b) & \text{Cov}(b,c) \\ \text{Cov}(c,a) & \text{Cov}(c,b) & \text{V}(c) \end{pmatrix} \qquad (3.51)$$

and is given by

$$\Sigma_{cov} = \begin{pmatrix} r_{aa} & r_{ab} & r_{ac} \\ r_{ba} & r_{bb} & r_{bc} \\ r_{ca} & r_{cb} & r_{cc} \end{pmatrix}^{-1} \qquad (3.52)$$

where

$$r_{ij} = -E\left[\frac{\partial^2 \ell}{\partial_i \partial_j}\right] , \quad i,j = a, b, c . \qquad (3.53)$$

Thus, to obtain Σ_{cov}, we first take the derivates of Equation (3.43) and then the expectations of the resulting expressions, ad indicated by Equation (3.53). On so doing, we get the following expressions for various $r_{i,j}$'s, i,j = a, b, c.

$$r_{aa} = \frac{1}{a} \sum_{i=1}^{n} (e^{-bt_{i-1}^c} - e^{-bt_i^c}) \qquad (3.54)$$

$$r_{ab} = r_{ba} = t_n^c \, e^{-bt_n^c} \qquad (3.55)$$

$$r_{ac} = r_{ca} = bt_n^c (\ell n t_n) e^{-bt_n^c} \qquad (3.56)$$

$$r_{bb} = a \sum_{i=1}^{n} \frac{(t_{i-1}^c + t_i^c)^2 \, e^{-b(t_{i-1}^c + t_i^c)}}{e^{-bt_{i-1}^c} - e^{-bt_i^c}} + at_n^{2c} e^{-bt_n^c} \qquad (3.57)$$

$$r_{bc} = r_{cb}$$

$$= a \sum_{i=1}^{n} \frac{\{t_i^c \ell n t_i e^{-bt_i^c} - t_{i-1}^c \ell n t_{i-1} e^{-bt_{i-1}^c}\} \cdot \{b(t_i^c e^{-bt_i^c} - t_{i-1}^c e^{-bt_{i-1}^c})\}}{e^{-bt_{i-1}^c} - e^{-bt_i^c}}$$

$$- (1-bt_i^c)(e^{-bt_{i-1}^c} - e^{-bt_i^c}) + a(1-bt_n^c) t_n^c (\ell n t_n) e^{-bt_n^c}$$

(3.58)

and

$$r_{cc} = ab \sum_{i=1}^{n} \frac{b(1-bt_i^c)^2 [t_i^c (\ell n t_i) e^{-bt_i^c} - t_{i-1}^c (\ell n t_{i-1}) e^{-bt_{i-1}^c}]^2}{e^{-bt_{i-1}^c} - e^{-bt_i^c}}$$

$$- ab \sum_{i=1}^{n} (1-bt_i^c) [t_i^c (\ell n t_i)^2 e^{-bt_i^c} - t_{i-1}^c (\ell n t_{i-1})^2 e^{-bt_{i-1}^c}]$$

$$- ab(1-bt_n^c) t_n^c (\ell n t_n)^2 e^{-bt_n} . \qquad (3.59)$$

The variance-covariance matrix Σ_{cov} is obtained by substituting the appropriate values from Equations (3.54) to (3.59) into Equation (3.53). Confidence bounds on the performance measures can then be computed by using the properties of a trivariate normal distribution.

3.5.2 Maximum Likelihood Estimation of Parameters When Data on Times Between Software Failures are Given

Sometimes failure data are given as a sequence of failure times $s_1, s_2, \ldots, s_n$ where s_k, $k = 1,2,\ldots,n$, represents the time of the kth failure. Using the joint density of $S_1, S_2, \ldots, S_n$, as given in Equation (3.35), the likelihood function of a, b, and c for given data $s_1, s_2, \ldots, s_n$ is

$$L(a,b,c|s_1,s_2,\ldots,s_n) = e^{-a(1-e^{-bs_n^c})} \prod_{k=1}^{n} (abc)\{s_k^{c-1} e^{-bs_k^c}\} . \quad (3.60)$$

As before, the maximum likelihood estimates are those values which maximize the likelihood function of Equation (3.60). Since maximizing the likelihood is equivalent to maximizing the log-likelihood function, we take the natural logarithm of Equation (3.60) and get

$$\ell(a,b,c|s_1,s_2,\ldots,s_n) \equiv \ell n\ L(a,b,c|s_1,s_2,s_n)$$

$$= n\ell na + n\ell nb + n\ell nc + (c-1)\sum_{k=1}^{n} \ell ns_k$$

$$- b\sum_{k=1}^{n} s_k^c - a(1-e^{-bs_n^c}) \ . \qquad (3.61)$$

Then, the mle's are those values $\hat{a}$, $\hat{b}$, $\hat{c}$ which satisfy the following equations:

$$\frac{\partial \ell}{\partial a} = 0 \ , \qquad (3.62)$$

$$\frac{\partial \ell}{\partial b} = 0 \ , \qquad (3.63)$$

$$\frac{\partial \ell}{\partial c} = 0 \ . \qquad (3.64)$$

On taking the derivatives of Equations (3.61), (3.62), (3.63), and (3.64), respectively,

$$n = a(1-e^{-bs_n^c}) \ , \qquad (3.65)$$

$$n = b\{\sum_{k=1}^{n} s_k^c + as_n^c e^{-bs_n^c}\} \ , \qquad (3.66)$$

and

$$n + c\sum_{k=1}^{n} \ell n s_k = bc\{\sum_{k=1}^{n} s_k^c \ell n s_k + a s_n^c (\ell n s_n) e^{-bs_n^c}\}. \quad (3.67)$$

The above simultaneous, non-linear equations can be solved numerically to get the maximum likelihood estimates $\hat{a}$, $\hat{b}$, and $\hat{c}$.

Since the joint distribution of $(S_1, S_2, \ldots, S_n)$ is an improper distribution, as discussed in section 2, the asymptotic properties of the mle's do not hold in this case.

3.6 ANALYSIS OF FAILURE DATA FROM A LARGE SCALE SOFTWARE SYSTEM

Failure data generated during formal testing of a large scale software system [THA76] was analyzed in section 2.8 using a two parameter non-homogeneous Poisson process model. In that analysis, the data from the first 9 of the 24 weeks of testing had to be dropped because during this period the system exhibited an increasing failure rate. The model developed in this section is capable of modelling an increasing/decreasing failure rate and will be employed to develop a model for the failure data over the entire 24 week testing period.

The number of failures per week for the four data sets are given in Table 3.1 and a plot for data set DS1 is shown in Figure 3.1. It is readily seen that the failure rate increases for about the first nine weeks and then decreases until the end of testing.

3.6.1 Estimation of Parameters

The data are given in the form of point (t_i, y_i), $i = 1,2,\ldots,24$, where t_i and y_i refer to time in weeks and y_i is the number of failures in week i. To esti-

TABLE 3.1. Software Failure Data From Thayer et al. [THAT76].

WEEK	DATA SET							
	DS1		DS2		DS3		DS4	
	# OF SPR's	CUMULATIVE	# OF SPR's	CUMULATIVE	# OF SPR's	CUMULATIVE	# OF SPR's	CUMULATIVE
1	67	67	69	69	79	79	77	77
2	106	173	111	180	144	223	139	216
3	97	270	99	279	141	364	139	355
4	129	399	133	412	199	563	195	550
5	77	476	87	499	143	706	133	683
6	143	619	156	655	233	939	220	903
7	116	735	131	786	202	1141	187	1090
8	131	866	141	927	203	1344	193	1283
9	187	1053	211	1138	316	1660	292	1575
10	203	1256	238	1376	369	2029	334	1909
11	136	1392	179	1555	298	2327	255	2164
12	183	1575	222	1777	398	2725	359	2523
13	47	1622	73	1850	115	2840	89	2612
14	46	1668	56	1906	83	2923	73	2685
15	71	1739	88	1994	150	3073	133	2818
16	54	1793	78	2072	142	3215	118	2936
17	57	1850	92	2164	172	3387	137	3073
18	80	1930	104	2268	202	3589	178	3251
19	64	1994	85	2353	168	3757	147	3398
20	27	2021	53	2406	89	3846	63	3461
21	42	2063	45	2451	87	3933	84	3545
22	55	2118	59	2510	111	4044	107	3652
23	62	2180	84	2594	253	4297	231	3883
24	11	2191	27	2621	70	4367	54	3937

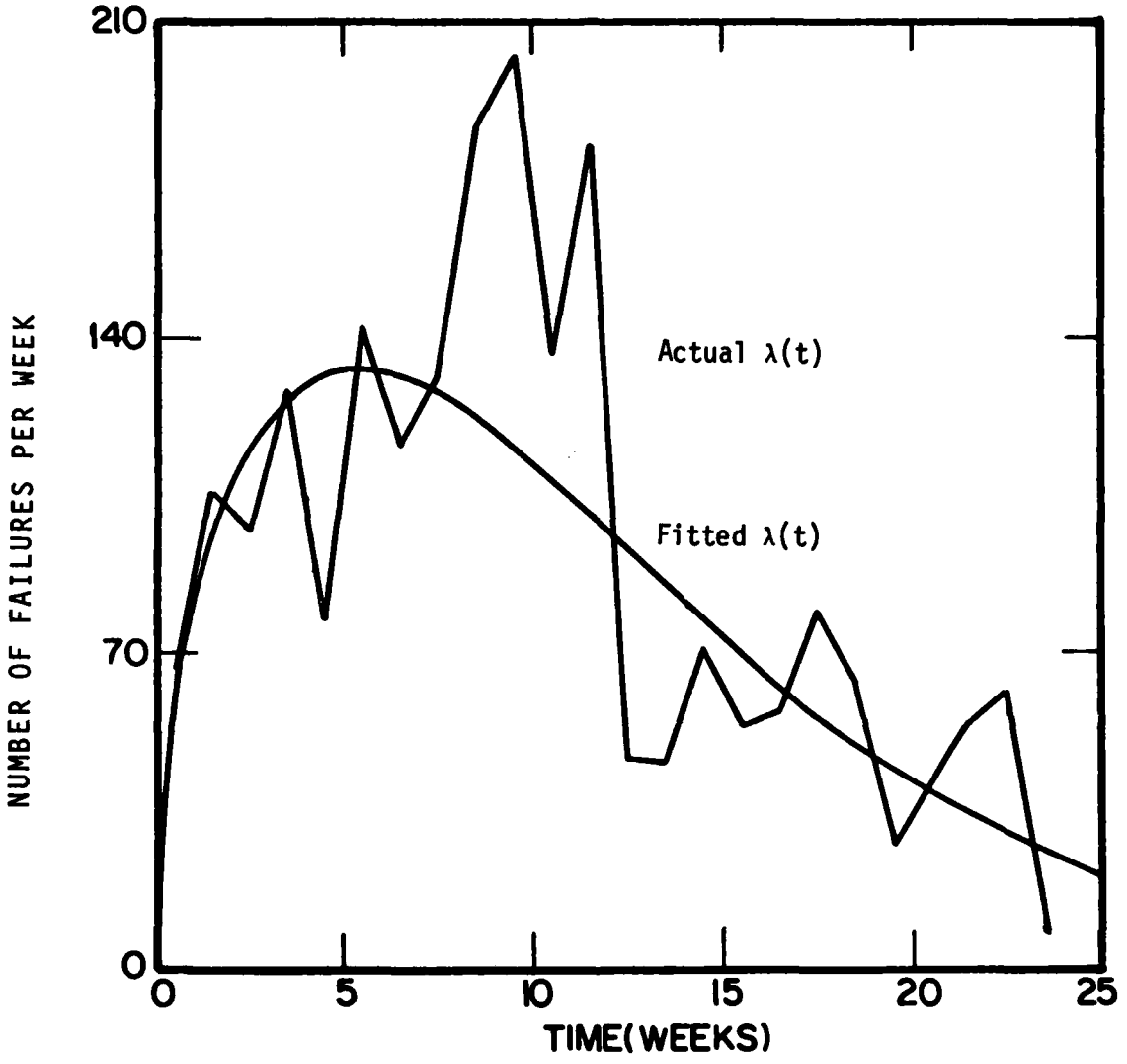


Fig. 3.1 A plot of the observed number of failures per week for Data Set DS1

mate the parameters a, b, and c, we use the method of section 3.5.1 and substitute the data values for each set in Equations (3.47), (3.48), and (3.49). The estimates $\hat{a}$, $\hat{b}$, and $\hat{c}$ for the four sets are given in Table 3.2. and the fitted mean value functions for the four data sets are as follows

$$\text{DS1:} \qquad \hat{m}(t) = 2352(1 - e^{-.0232t^{1.494}})$$

$$\text{DS2:} \qquad \hat{m}(t) = 2873(1 - e^{-.0182t^{1.540}})$$

$$\text{DS3:} \qquad \hat{m}(t) = 5182(1 - e^{-.0135t^{1.547}})$$

$$\text{DS4:} \qquad \hat{m}(t) = 4657(1 - e^{-.0156t^{1.505}})$$

A plot of the cumulative number of observed software failures is given in Figure 3.2 and the expected cumulative number of failures ($\hat{m}(t)$) for each data set are shown in Figure 3.3.

TABLE 3.2

A SUMMARY OF DATA ANALYSIS FOR DS1-DS4

Quantity \ Data Set	DS1	DS2	DS3	DS4
$\hat{a}$	2352	2873	5182	4657
$\hat{b}$	0.0232	0.0182	0.0135	0.0156
$\hat{c}$	1.494	1.540	1.547	1.505
$\sqrt{\widehat{Var}(\hat{a})}$	55.4	65.3	118.0	110.5
$\sqrt{\widehat{Var}(\hat{b})}$	0.00188	0.00143	0.00086	0.00101
$\sqrt{Var(\hat{c})}$	0.0354	0.0345	0.0297	0.0304
$\hat{\rho}_{a,b}$	0.184	0.224	0.286	0.273
$\hat{\rho}_{a,c}$	-0.320	-0.391	-0.584	-0.579
$\hat{\rho}_{b,c}$	-0.916	-0.914	-0.876	-0.868
Number of Errors Detected (Observed) During Operational Demonstration Period	198	263	540	475
Estimate of the Number of Errors to be Detected During Operational Demonstration Period	161	252	812	717

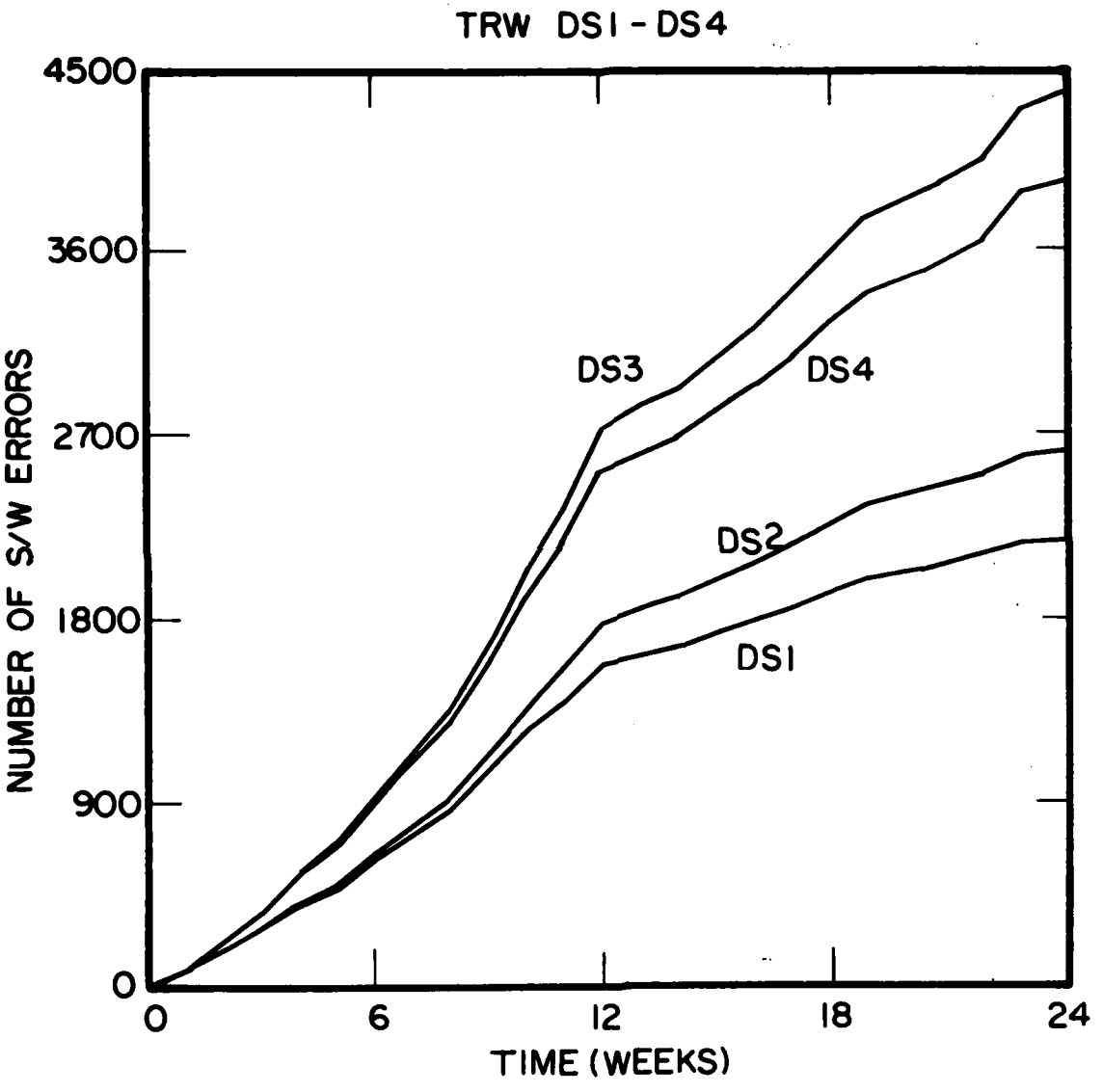


FIGURE 3.2. Plots of the Cumulative Number of Software Failures for DS1 to DS4.

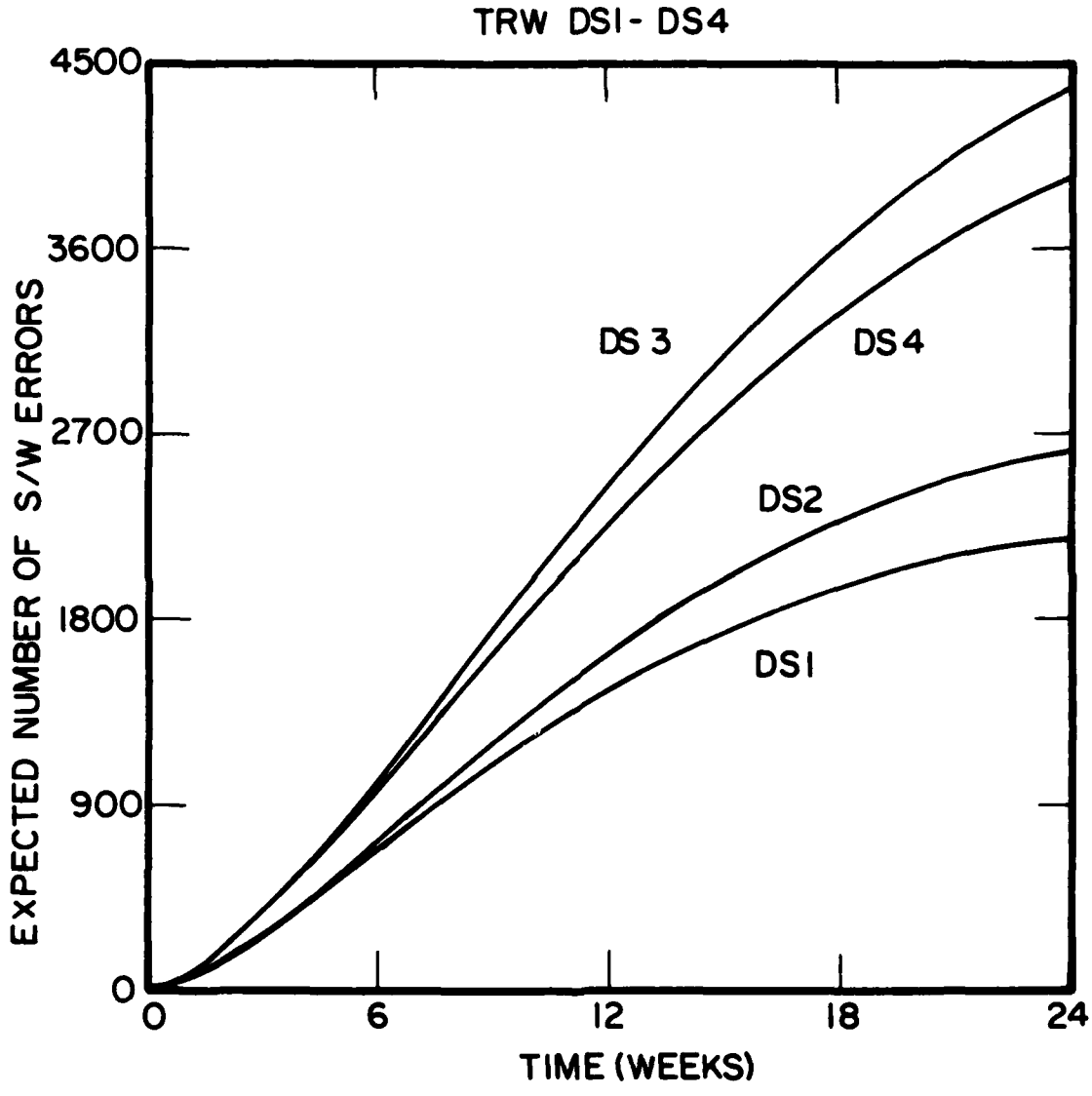


FIGURE 3.3. Plots of the Expected Cumulative Number of Software Failures ($\hat{m}(t)$) for DS1 to DS4.

3.6.2 Confidence Bounds

By using normal approximation to a Poisson in Equation (3.22) we compute the 90% confidence bounds for the N(t) process. The estimated mean value function and 90% bounds for data set DS1 for the N(t) process are plotted in Figure 3.4. From this figure we see that most of the observed points fall within 90% bounds implying that the model described in Section 3.2 fits the entire history of software errors very well. The number of remaining errors at t = 24 (weeks), given that 2191 errors were found by this time, is estimated from Equation (3.25) and we have

$$E[\overline{N}(24) \,|\, N(24) = 2191] = 2352e^{-.0232(24)^{1.494}} = 161.9.$$

Note that a total of 198 errors were detected during the one year period of operational demonstration so that the predicted number is close to the actual value. The variance-covariance matrix is obtained from Equation (3.51) and is

$$\hat{\Sigma}_{cov} = \begin{bmatrix} 3067 & 0.0191 & -0.627 \\ 0.0191 & 3.53\text{x}10^{-6} & -6.09\text{x}10^{-5} \\ -0.627 & -6.09\text{x}10^{-5} & 1.25\text{x}10^{-3} \end{bmatrix}$$

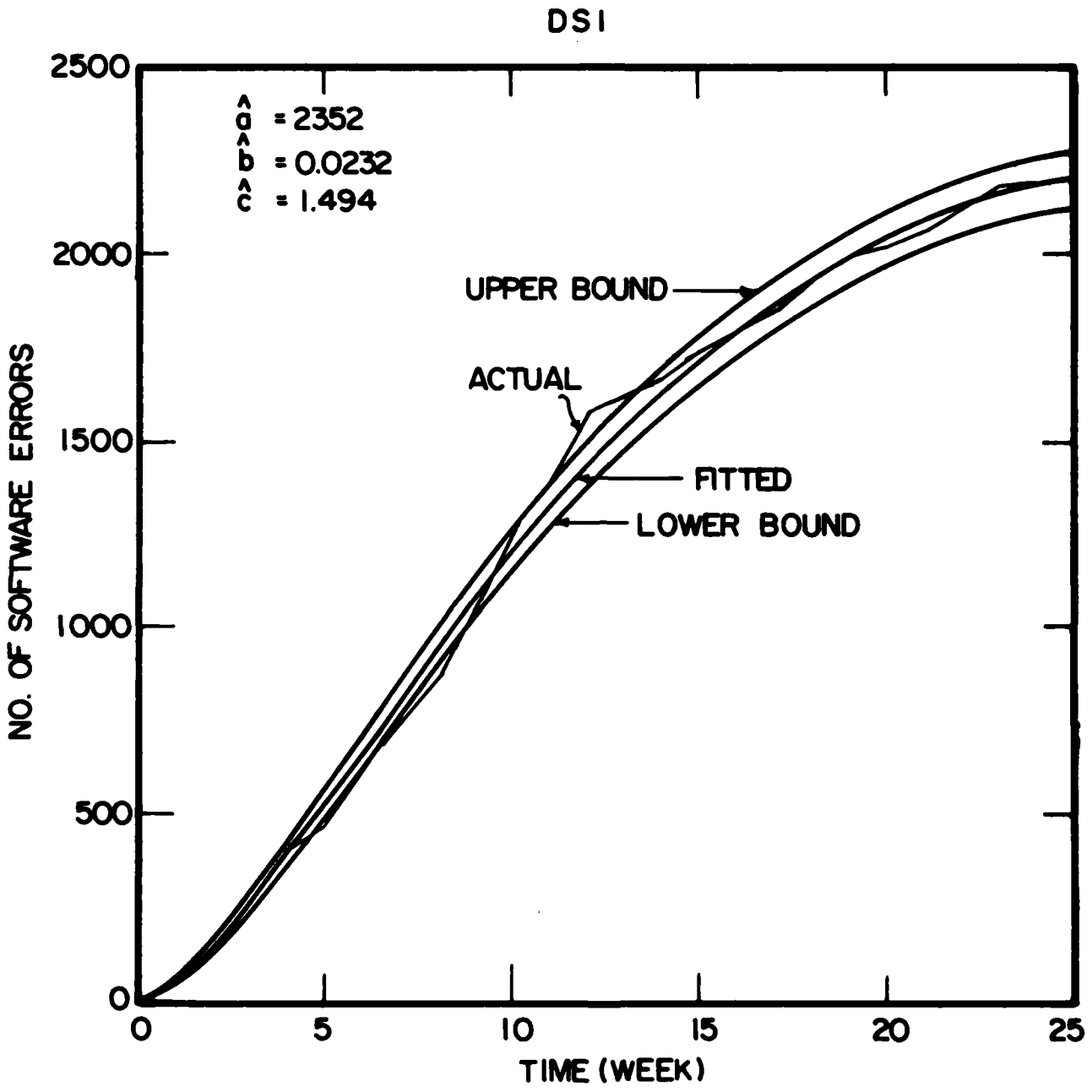


FIGURE 3.4. Estimated Mean Value Function and 90% Confidence Bounds for the N(t) Process (DS1).

From this matrix we obtain the estimated standard deviations and the appropriate correlation coefficients for $\hat{a}$, $\hat{b}$, and $\hat{c}$. These values for data sets DS1 to DS4 are also shown in Table 3.2. By using the above variance-covariance matrix we can also obtain $100(1-\alpha)\%$ confidence bounds for $E\bar{N}(t)$ which are given by

$$\{\hat{f}(a,b,c) \pm t_{n-3;\alpha/2}\sqrt{\hat{Var}(\hat{f}(a,b,c,))}\}$$

where

$$Var\{\hat{f}(a,b,c)\} = \left(\frac{\partial f}{\partial a}\ \frac{\partial f}{\partial b}\ \frac{\partial f}{\partial c}\right)\Sigma_{cov}\left(\begin{array}{c}\frac{\partial f}{\partial a}\\ \frac{\partial f}{\partial b}\\ \frac{\partial f}{\partial c}\end{array}\right)\Bigg|_{\substack{a=\hat{a}\\ b=\hat{b}\\ c=\hat{c}}}$$

For this case we have

$$\frac{\partial f}{\partial a} = e^{-bt^c}$$

$$\frac{\partial f}{\partial b} = -at^c e^{-bt^c}$$

$$\frac{\partial f}{\partial c} = -abt^c(\ell nt)e^{-bt^c}$$

The 90% confidence bounds for $E\bar{N}(t)$ for data set DS1 are computed from the above equations and are shown in Figure 3.5. Also shown is a plot of the actual number of remaining errors during the 24 week period. From this figure we see that the actual errors fall within the 90% bounds.

Similarly, by setting

$$\hat{f}(a,b,c) \equiv \hat{R}_{X_k|S_{k-1}}(x|s)$$
$$= e^{-\hat{a}\{e^{-\hat{b}s^{\hat{c}}} - e^{-\hat{b}(s+x)^{\hat{c}}}\}}$$

we can estimate the software reliability for given debugging time s.

The $100(1-\alpha)\%$ confidence bounds on $R_{X_k|S_{k-1}}(x|s)$ can be obtained as for $E\bar{N}(t)$. The reliability plots and 90% confidence bounds for DS1 are shown in Figure 3.6.

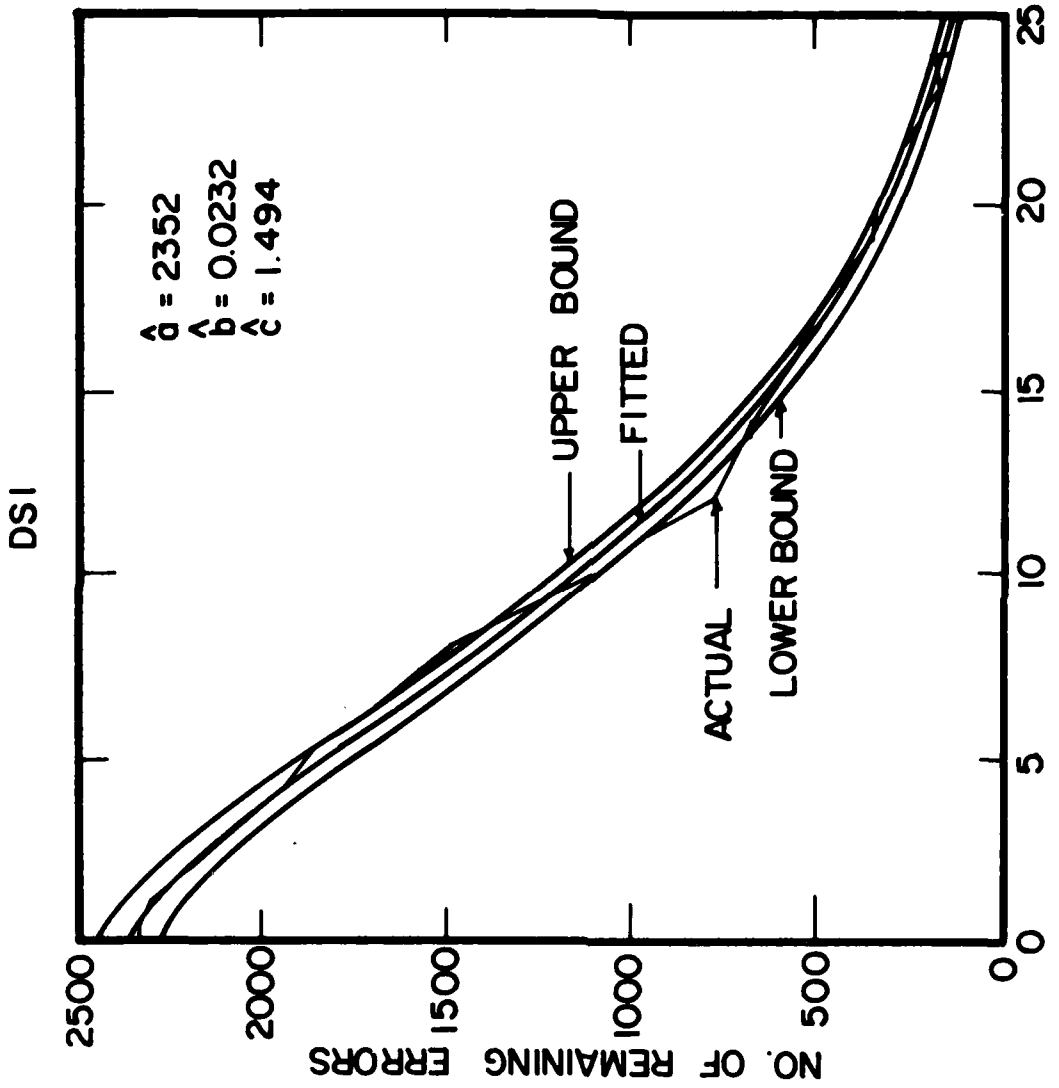


FIGURE 3.5. Estimated 90% Confidence Bounds for $E(\bar{N}(24))$ and values of $E\bar{N}(t)$ (DS1).

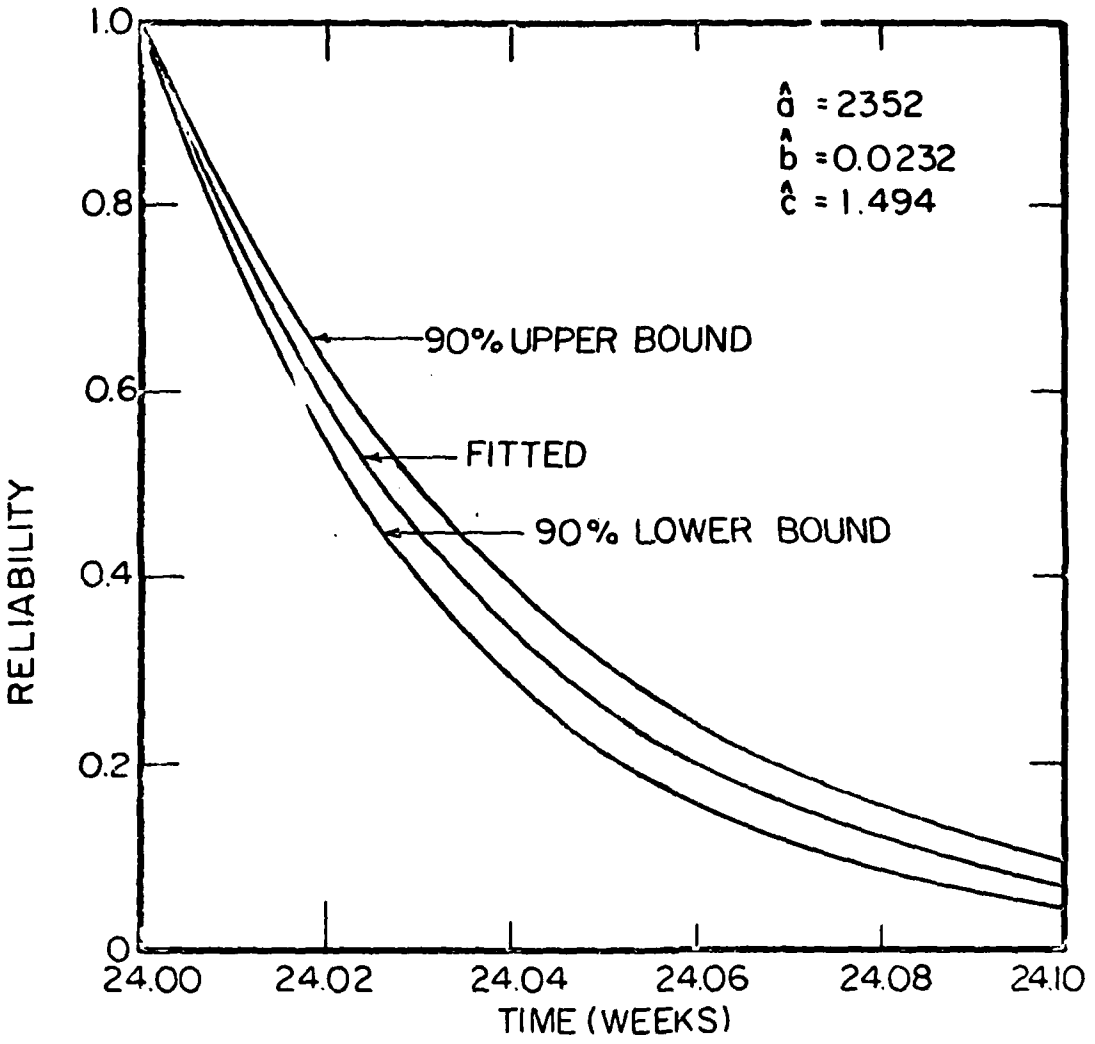


FIGURE 3.6. Reliability Function and 90% Confidence Bounds (DS1).

3.7 ANALYSIS OF FAILURE DATA FROM NAVAL TACTICAL DATA SYSTEM (NTDS)

Failure data from NTDS were analyzed in Section 2.7 using a two parameter NHPP model. In this section we reanalyze the same data by using the three parameter NHPP model of Section 3.2. (For details of the system and data set, see Section 2.7.) Data analysis using the Newton-Raphson method for solving the likelihood estimates of a, b, and c based on the first 26 failures of Table 2.1, we get $\hat{a} = 27.2$, $\hat{b} = 0.000783$, and $\hat{c} = 1.50$ so that

$$\hat{m}(t) = \hat{a}(1-e^{-\hat{b}t^{\hat{c}}})$$

$$= 27.2(1-e^{-0.000783t^{1.5}}) .$$

The bounds of the N(t) process can be obtained by using normal approximation to a Poisson distribution of Equation (3.22). The estimated mean value function and 90% bounds of the N(t) process for this data set are shown in Figure 3.7. Also shown is a plot of the actual number of errors detected by time t. From this figure we see that all the data points fall within the 90% bounds. We can estimate the expected number of errors remaining

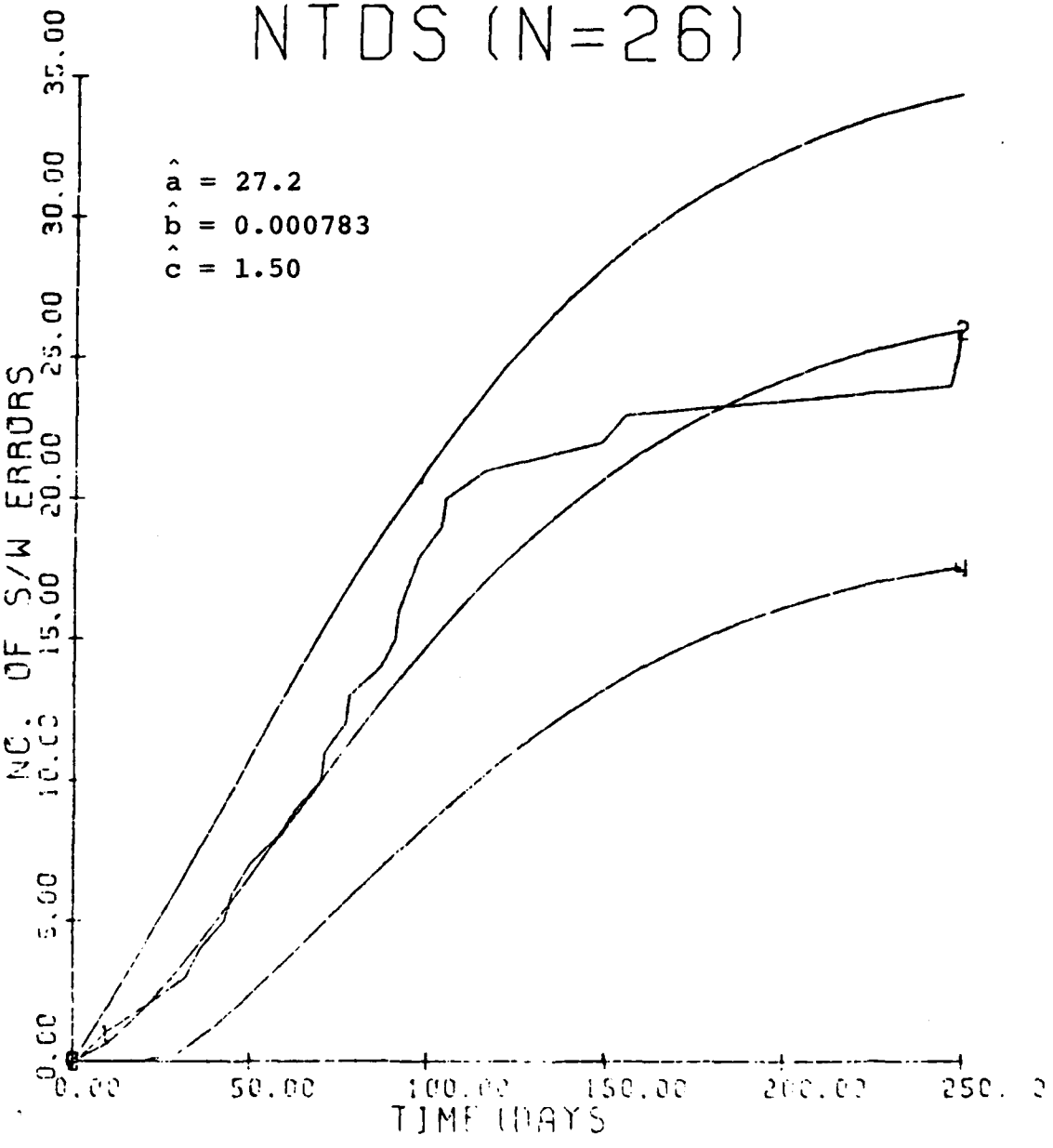


FIGURE 3.7. Estimated Mean Value Function and 90% Confidence Bounds for the N(t) Process (Data Set NTDS).

at time t by substituting the mle's in Equation (3.25), i.e.,

$$\hat{E\bar{N}}(t) = \hat{a}e^{-\hat{b}t^{\hat{c}}}$$

or

$$\hat{E\bar{N}}(t) = 27.2e^{-0.000783t^{1.5}} .$$

Thus for t = 250,

$$E\bar{N}(250) = 1.23 .$$

That is, we can expect one more error remaining at t = 250 (days). The conditional reliability of the time to the next (27th) failure, given S_{26} = 250, is computed as

$$\hat{R}_{X_{27}|S_{26}}(x|250) = e^{-\hat{a}\{e^{-\hat{b}(250^{\hat{c}})}-e^{-\hat{b}(250+x)^{\hat{c}}}\}}$$

$$= e^{-27.2\{0.0453-e^{-0.000783(250+x)^{1.5}}\}}.$$

For the values of x = 10, 20, and 50 (days) the reliability values are 0.81, 0.68, and 0.46, respectively.

SECTION 4

OPTIMUM SOFTWARE RELEASE TIME

4.1 INTRODUCTION

An important objective of developing the models in Sections 2 and 3 was to provide an analytical framework for estimating software performance measures which are needed for making various decisions. An important decision of practical concern is the determination of the time when testing can stop and the system be considered ready for release, that is, the determination of the software release time.

The operational performance of a software system is to a large extent dependent on the time spent in testing. The longer the testing phase, the better the performance. Also, the cost of fixing an error is generally much less during testing than during operation. However, the time spent in testing delays the release of the system for operational use and incurs additional cost. This suggests a reduction in test time and an early release of the system. In this section, we consider these conflicting objectives in the determination of the optimum release time.

In Section 4.2 we consider the release time problem based on a reliability criterion using the model of Section 2. Cost based optimum release time policies are developed in Sections 4.3 and 4.4 when the failure phenomenon follows a non-homogeneous Poisson process. The policy in Section 4.3 uses the model of Section 2 while the policy in Section 4.4 is for the failure model of Section 3.

4.2 SOFTWARE RELEASE TIME BASED ON RELIABILITY CRITERION

For a non-homogeneous Poisson process failure model, the conditional reliability at operational time x, given that the testing has proceeded for S = s time units, is given by

$$R_{X|S}(x|s) \equiv R = \exp[-a(e^{-bs} - e^{-b(s+x)})] \qquad (4.1)$$

or

$$R = \exp[-m(x)e^{-bs}], \qquad (4.2)$$

where

$$m(x) = a(1 - e^{-bx}) \ . \qquad (4.3)$$

One commonly used criterion is to stop testing when the predicted reliability at a specified time x equals some given value. Then the problem reduces to solving (4.2) to find the value of s that satisfies this criterion.

Taking the logarithm of both sides of (4.2) and rearranging yields

$$s = (1/b)[\ell n\ m(x) - \ell n\ \ell n\ 1/R] \qquad (4.4)$$

for the software system under test. In (4.3) and (4.4), a and b are estimated from previous data and R and x are the specified values. Therefore, the required testing time s can be easily determined.

For illustration purposes, consider the failure data DS1 discussed in Section 2.8. For this data set a = 1348 and b = 0.124. Suppose it is desired that the testing be continued until the operational reliability at x = 0.1 equals 0.70. From (4.4),

$$s = (1/0.124)\{\ell n[1348(1-e^{-0.124(0.1)})]$$

$$- \ell n\ \ell n(1/0.7)\},$$

or s = 31 weeks.

In other words, 31 weeks of testing will be needed before the system can be released to assure the desired reliability.

To see the effect of s on R(x|s), plots of reliability versus bs for m(x) = 5(5)50 are shown in Figure 4.1. We note that, as the testing time s is increased, while keeping x, and hence m(x), constant, R(x|s) in-

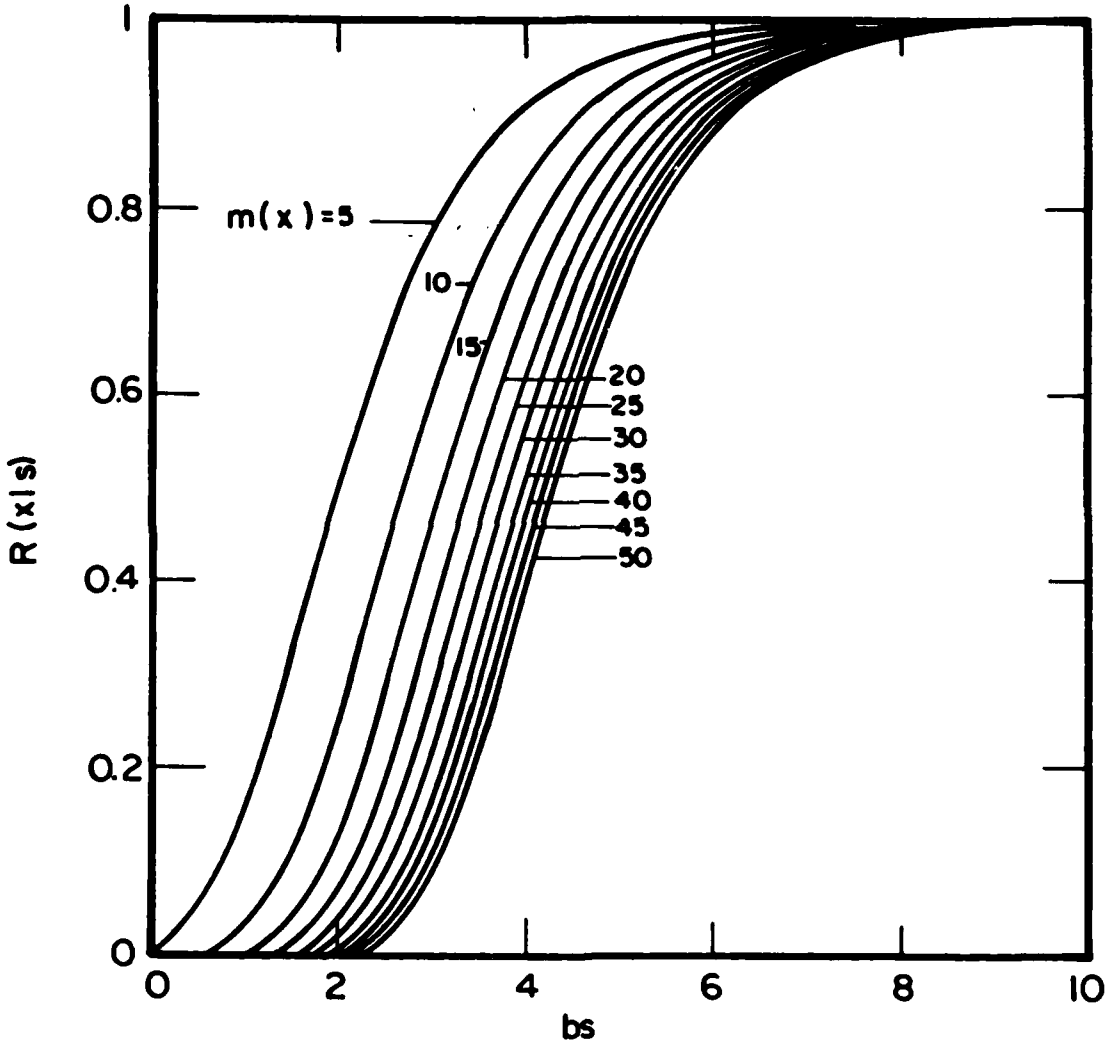


Fig. 4.1 Plots of reliability versus bs for m(x) = 5(5)50.

creases very rapidly to approximately 0.95. After that, the increase in reliability is very slow, which indicates that a long testing time is required to get a highly reliable software system.

4.3 OPTIMUM RELEASE TIME BASED ON COST CRITERION (MODEL OF SECTION 2)

To determine the optimal policy, we first develop a cost model and then solve it to get the desired result.

4.3.1 Cost Model and Optimal Policy

Let

c_1 = cost of fixing an error during testing,

c_2 = cost of fixing an error during operation ($c_2 > c_1$),

c_3 = cost of testing per unit time,

t = software life-cycle length, and

T = software release time (same as testing time).

Since $m(t)$ represents the expected number of errors during $(0,t)$, the expected costs of fixing errors during the testing and the operational phases are $c_1 m(T)$ and $c_2[m(t) - m(T)]$, respectively. Further, the testing cost during T is $c_3 T$.

Combining the above costs, the total expected cost is given by

$$C(T,t) \equiv C(T) = c_1 m(T) + c_2[m(t)-m(T)] + c_3 T. \quad (4.5)$$

These costs are also shown in Figure 4.2.

Our objective is to find the optimum value T^* that minimizes (4.5). Differentiating (4.5) with respect to T, we get

$$dC(T)/dT = c_1 m'(T) - c_2 m'(T) + c_3 . \quad (4.6)$$

Equating the right-hand side of (4.6) with zero and noting that $\lambda(T) \equiv m'(T)$, we get

$$\lambda(T) = c_3/(c_2 - c_1) , \quad (4.7)$$

where

$$\lambda(T) = abe^{-bT} . \quad (4.8)$$

Note that $\lambda(T)$ is a monotonically decreasing function of T and $\lambda(0) = ab$. If $ab \leq c_2/(c_2 - c_1)$, (4.7) has no feasible solution and, for $T \geq 0$, $dC(T)/dT > 0$ (see

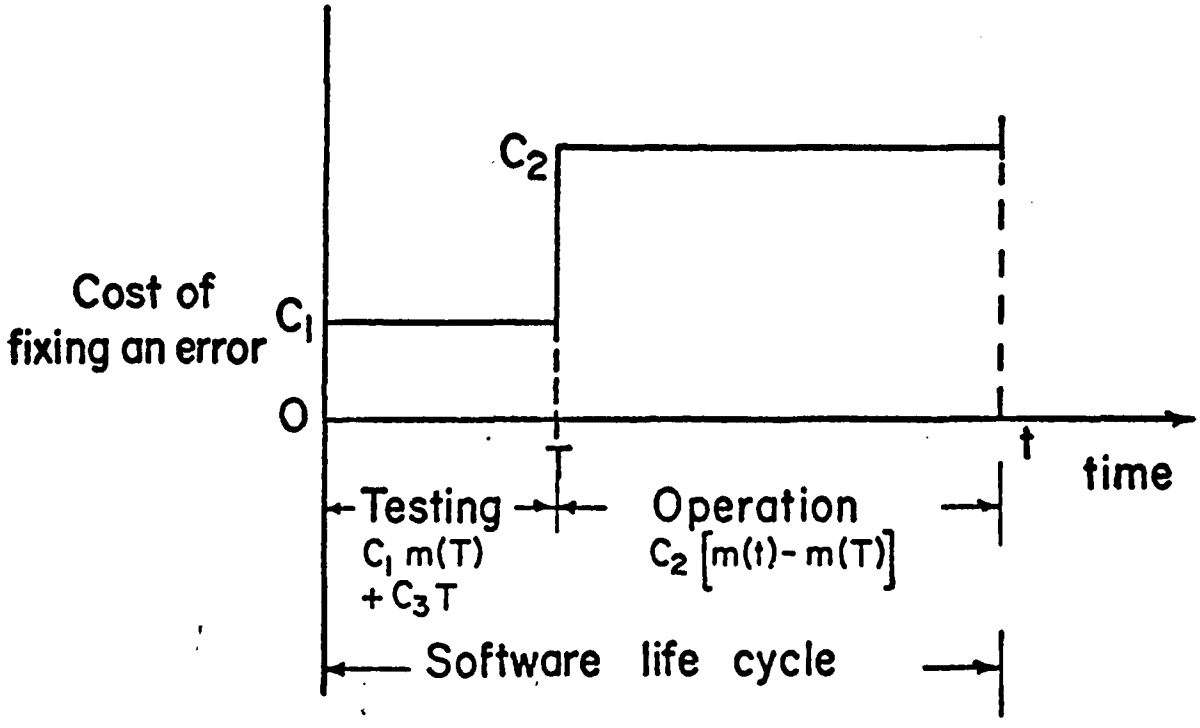


FIG. 4.2. COST COMPONENTS

Figure 4.3. Hence, for this case, the minimum of C(T) is at T = 0; that is, $T^* = 0$.

Now, if $ab > c_3/(c_2 - c_1)$, there exists a unique feasible solution of (4.7) given by (see Figure 4.3)

$$T_0 = (1/b)\ n\left[\frac{ab(c_2 - c_1)}{c_3}\right] . \qquad (4.9)$$

Since $dC(T)/dT < 0$ for $0 < T < T_0$ and $dC(T)/dt > 0$ for $T > T_0$, the minimum of C(T) is at $T = T_0$ for $T_0 \leq T$ and at $T = t$ for $T_0 > t$. These can be summarized as follows.

Theorem 4.1. (i) If $ab > c_3/(c_2 - c_1)$, then there exists a unique feasible solution of (4.7) and the optimum release time is

$$T^* = \min\{T_0, t\} ,$$

where T_0 is given by (4.9).

(ii) If $ab \leq c_3/(c_2 - c_1)$, then $T^* = 0$.

It should be noted that, if the minimum expected cost exceeds the operational benefit to be ained, no testing should be undertaken.

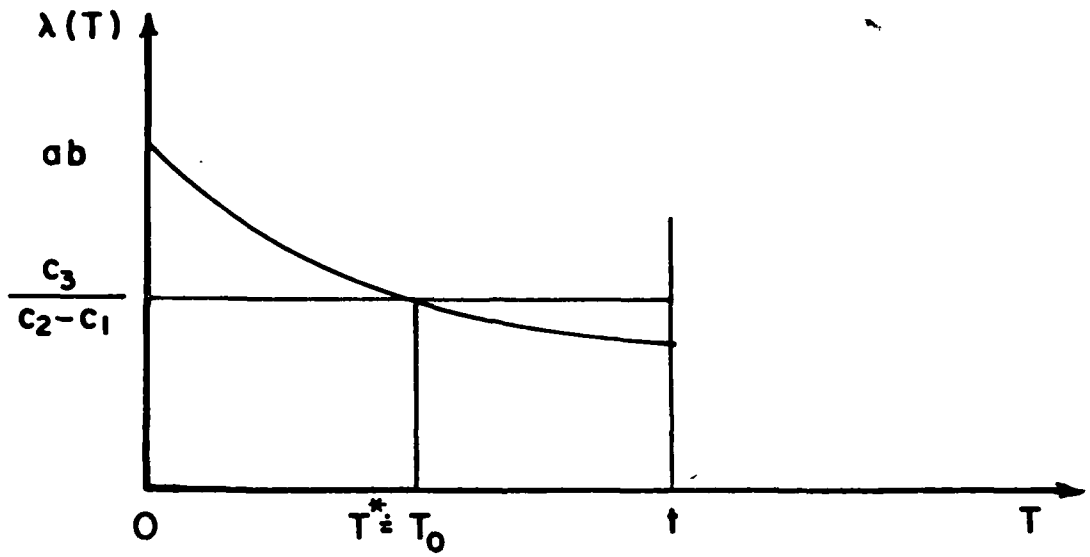


FIG. 4.3. PLOTS OF λ(T) VERSUS T.

To illustrate the above results, consider the data set mentioned in Section 4.2. Here a = 1348 and b = 0.124. Let $c_1 = 1$, $c_2 = 5$, $c_3 = 100$, and t = 100.

Then ab = 1348(0.124) = 167 and

$$c_3/(c_2 - c_1) = 25 .$$

Since ab > $c_3/(c_2 - c_1)$, the optimum release time

$$T^* = \min\{(1/0.124)\ln(167/25), 100\}$$

or

$$T^* = \min\{15.3, 100\} = 15.3 .$$

Hence, the optimum solution for this case is to allocate 15.3 weeks for testing and 84.7 weeks for operation. The cost associated with this policy will be $C(T^*) = 3687$.

4.3.2 Sensitivity Analysis of T^*

Now we investigate the effects of the parameters a, b, and $c_r \equiv c_3/(c_2 - c_1)$ on the optimum release time.

First, from Theorem 4.1, we note that T^* equals 0, t, or T_0. Since $T^* = 0$ and $T^* = t$ are degenerate cases, we shall consider only the case when $T^* = T_0$.

From (4.9), we see that T_0 increases logarithmically with a and decreases logarithmically with c_r as others are kept constant. Next, the first and second derivatives of T_0 with respect to b indicate that T is a concave function of b with maximum at $b_0 \equiv ec_r/a$, and the maximum value of T_0 is $1/b_0$.

In practice, for a given software system, the value of a prior to testing is fixed and one may be interested in the joint effect of b and c_r on T_0. The value of b can be affected by an appropriate selection of testing strategies and techniques. For the data set discussed earlier, a = 1348. For this case, contours of T_0 in the b-c_r plane are shown in Figure 4.4. Also shown is the optimum value of T corresponding to the above numerical example. This diagram can also be used to determine the value of b if T_0 is fixed due to some other considerations. Thus, if $c_r = 25$, and $T_0 = 15$, we need b = 0.13. If, however, $T_0 = 10$, b must be 0.265.

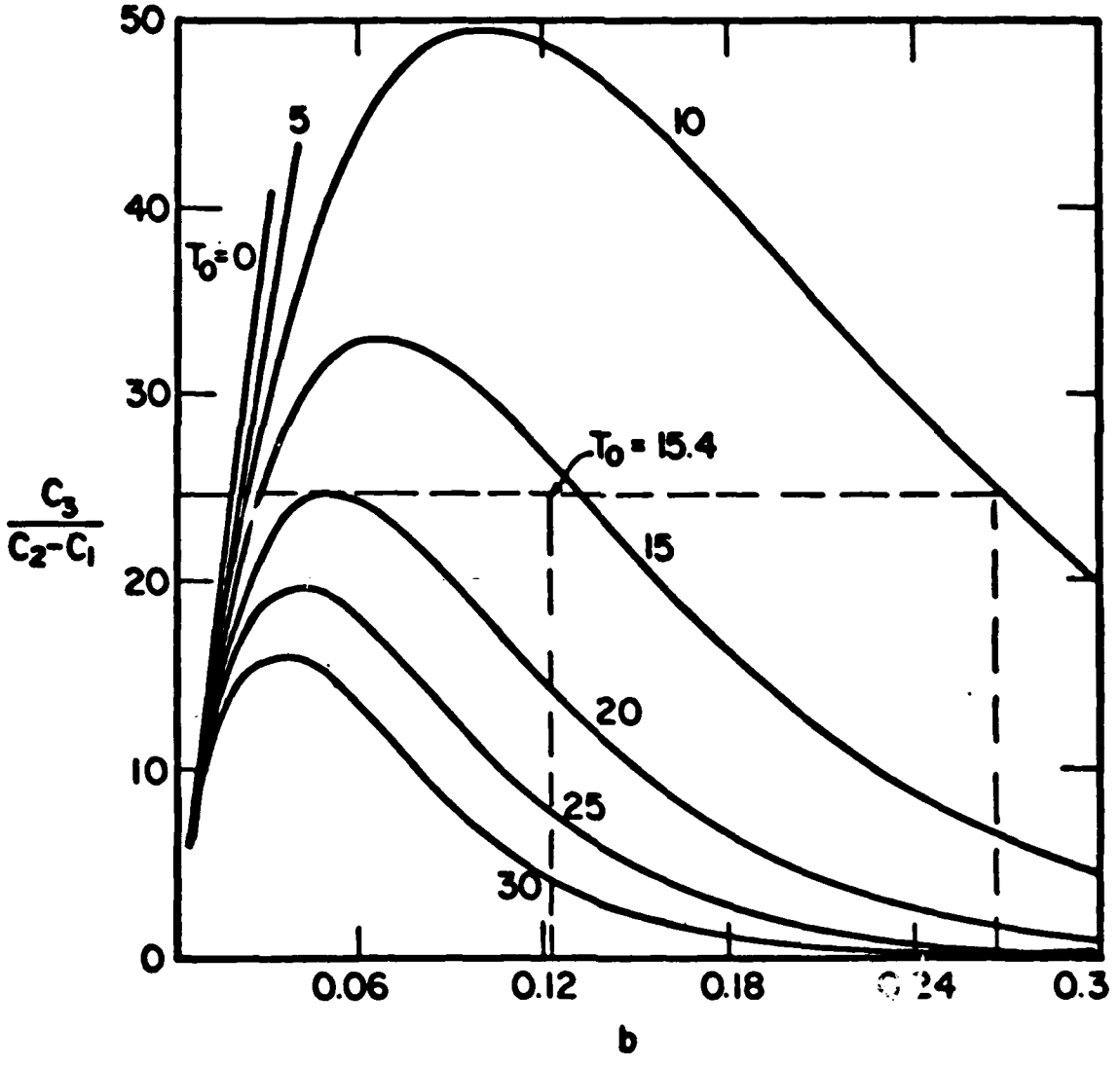


Fig. 4.4 Contours of T_0 in the b, $c_3/(c_2 - c_1)$ plane (a = 1348).

4.4 OPTIMUM RELEASE TIME BASED ON COST CRITERION (MODEL OF SECTION 3)

4.4.1 Cost Model and Optimal Policy

The cost model for this case is similar to that given in Equation (4.5) and is (quantities are as defined in Section 4.3.1)

$$c(T,t) = c_1 M(T) + c_2[M(t) - M(T)] + c_3 T \qquad (4.10)$$

where

$$M(x) = a(1 - e^{-bx^c}) \ . \qquad (4.11)$$

On differentiating (4.10) with respect to T and equating the result to zero, we get

$$\lambda(T) \equiv M'(T) = \frac{c_3}{c_2 - c_1} \ , \qquad (4.12)$$

where

$$\lambda(T) = \alpha T^{\gamma-1} \cdot e^{-\beta T^{\gamma}} \qquad (4.13)$$

$$\alpha = abc$$

$$\beta = b$$

$$\gamma = c$$

To solve (4.12) for T, we consider three cases depending on the value of γ, viz $\gamma > 1$, $\gamma = 1$, and $0 < \gamma < 1$.

Case When $\gamma > 1$

For this case, the failure distribution has an increasing failure rate followed by a decreasing failure rate. Then we see from (4.13) that $\Lambda(T)$ is a unimodal function of T with $\Lambda(0) = \Lambda(\infty) = 0$. Also, its maximum $\Lambda(T_m)$ occurs at T_m where

$$T_m = \left(\frac{\gamma-1}{\beta\gamma}\right)^{1/\gamma} \tag{4.14}$$

and

$$\Lambda(T_m) = \alpha\left(\frac{\gamma-1}{\beta\gamma e}\right)^{(\gamma-1)/\gamma}. \tag{4.15}$$

If $\Lambda(T_m) < \frac{c_3}{c_2 - c_1}$ and $\Lambda(T) < \frac{c_3}{(c_2 - c_1)}$ for $T \geq 0$, then it is easy to see that Equation (4.12) has no feasible solution for T. Therefore, for $T \geq 0$, $\frac{dC(T,t)}{dt} > 0$, and the minimum of $C(T;t)$ is at $T = 0$. In other words, if $\Lambda(T_m) < \frac{c_3}{c_2 - c_1}$ and $\Lambda(T) < \frac{c_3}{c_2 - c_1}$, then $T^* = 0$. This is shown graphically in Figure 4.5.

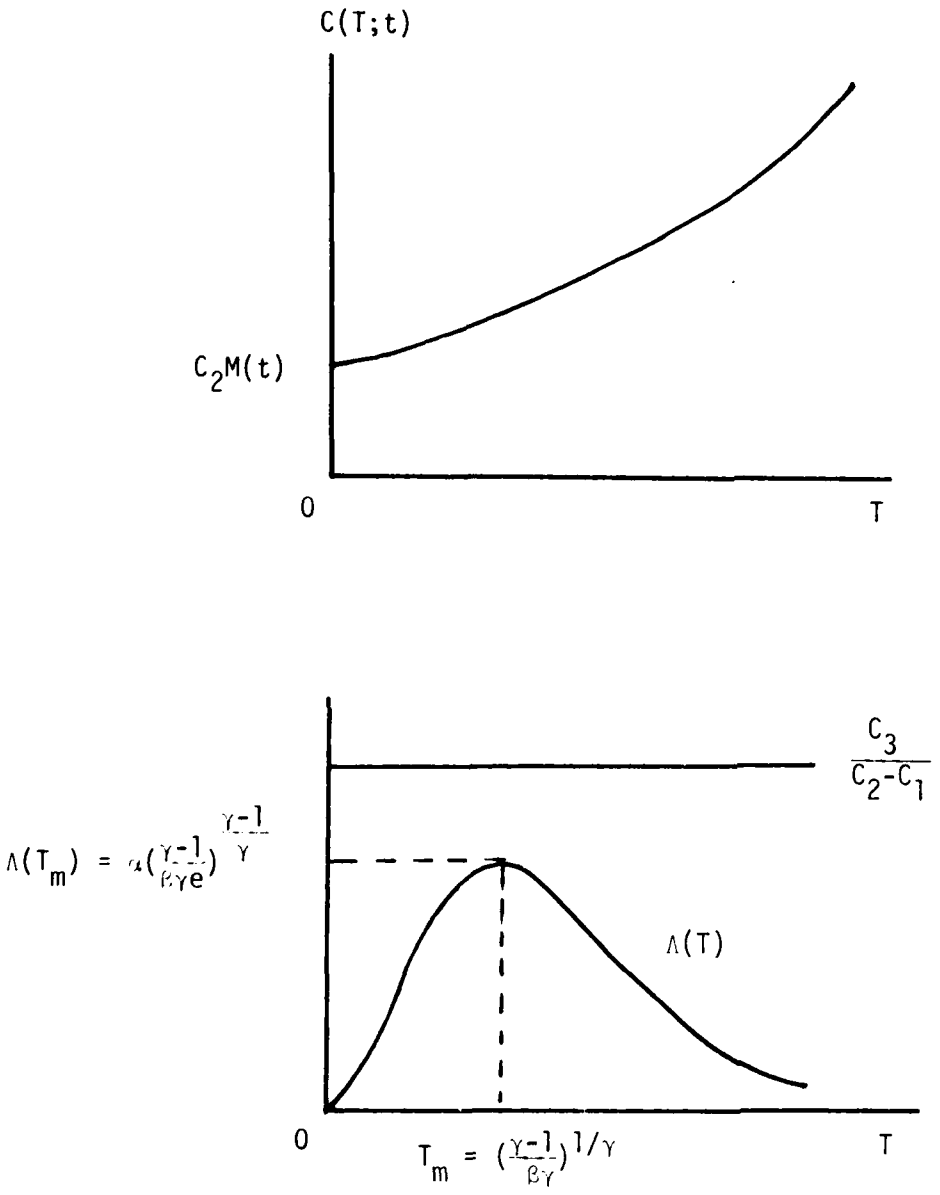


Fig. 4.5. Plots of C(T;t) and $\Lambda(T)$ for $\gamma > 1$ and $\alpha(\frac{\gamma-1}{\beta\gamma e})^{(\frac{\gamma-1}{\gamma})} < \frac{C_3}{C_2-C_1}$.

If $\Lambda(T_m) = \frac{c_3}{(c_2 - c_1)}$, $\frac{d}{dT} C(T;t) = 0$ for $T = T_m$, then $\frac{d}{dT} C(T;t) > 0$ for $0 < T < T_m$ and for $T > T_m$. Then Equation (4.12) has a unique feasible solution $T = T_m$. However, $T = T_m$ is an inflection point of $C(T;t)$ for this case. Therefore, the minimum of $C(T;t)$ is at $T = 0$, i.e., $T^* = 0$ as seen in Figure 4.6.

If, however, $\Lambda(T_m) > \frac{c_3}{c_2 - c_1}$, there exist two feasible solutions $T = T_1$ and $T = T_2$, $0 < T_1 < T_2 < \infty$, which are the two positive roots of Equation (4.12). Also,

$$\Lambda(T) < \frac{c_3}{c_2 - c_1}, \quad 0 \leq T < T_1, \; T > T_2$$

and

$$\Lambda(T) > \frac{c_3}{c_2 - c_1}, \quad T_1 < T < T_2 .$$

For this case, T_1 and T_2 can be obtained by solving Equation (4.12) numerically. It should also be pointed out that $\frac{dC(T;t)}{dT} > 0$ for $0 \leq T < T_1$, $T > T_2$, and $\frac{dC(T;t)}{dT} < 0$ for $T_1 < T < T_2$. In this case, we consider the minimum of $C(T;t)$ for the following three cases (see Figure 4.7).

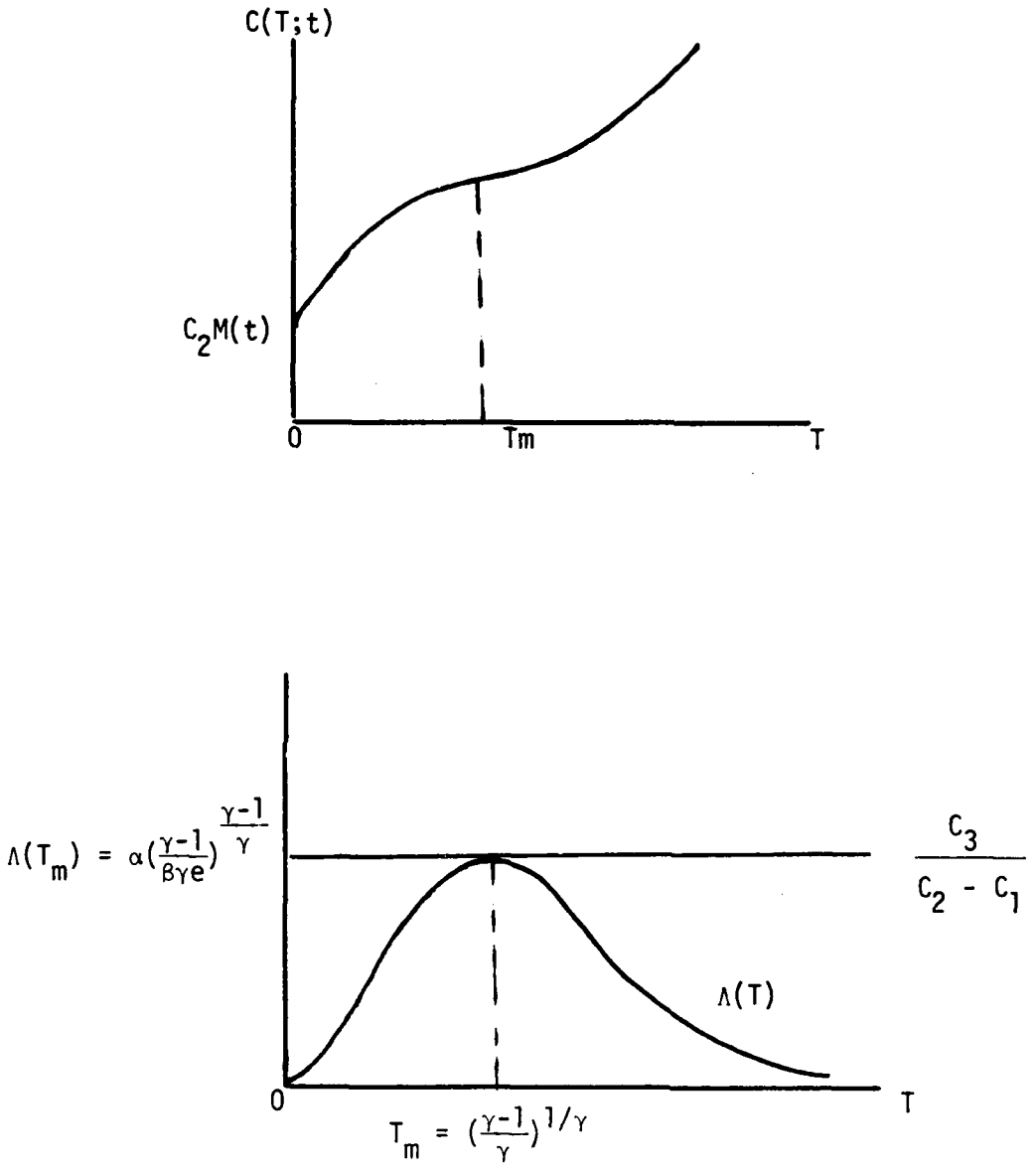


Fig. 4.6. Plots of $C(T;t)$ and $\Lambda(T)$ for $\gamma > 1$ and $\alpha(\frac{\gamma-1}{\beta\gamma e})^{(\frac{\gamma-1}{\gamma})} = \frac{C_3}{C_2-C_1}$.

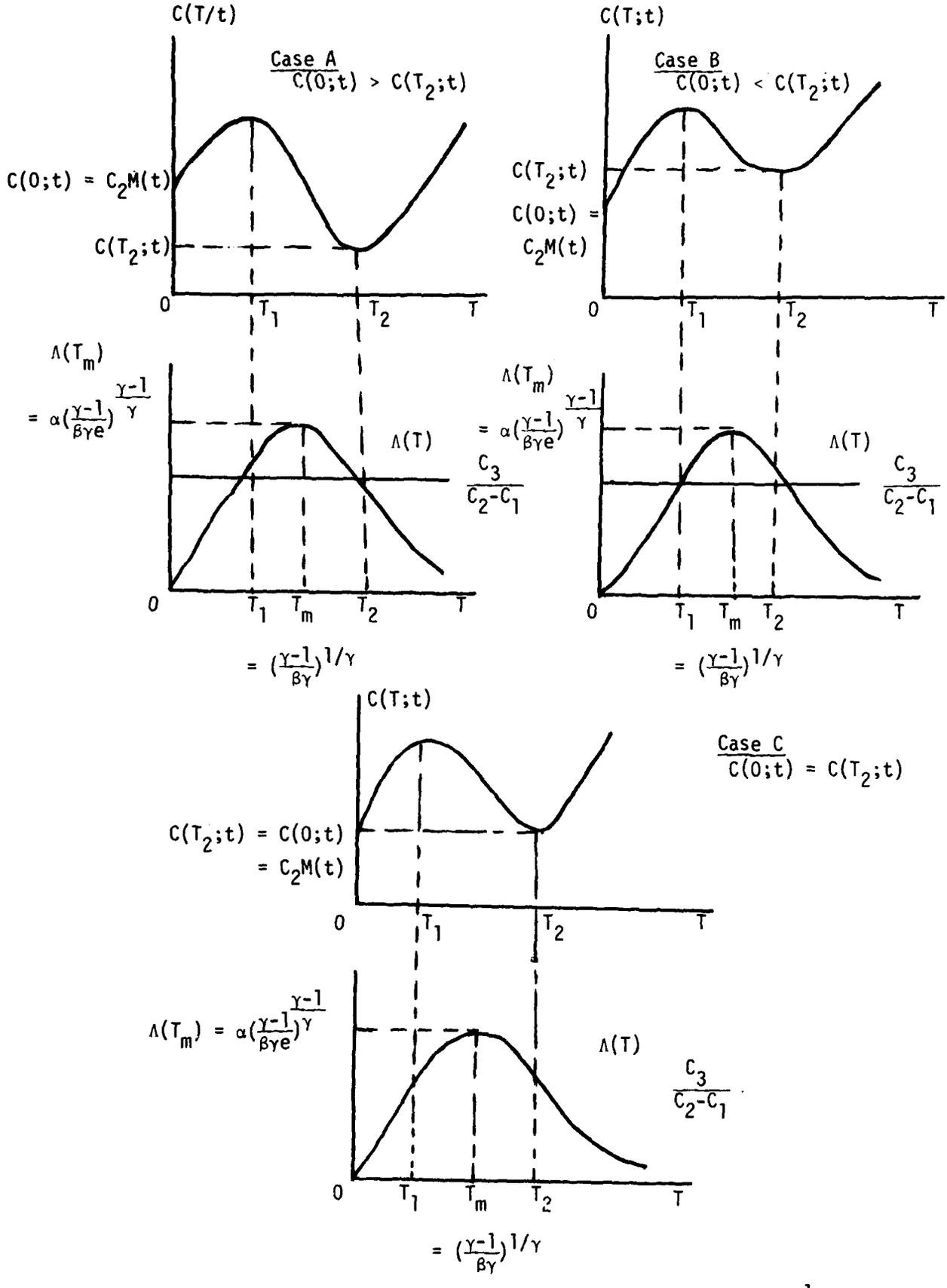


Fig. 4.7. Plots of $C(T;t)$ and $\Lambda(T)$ for $\gamma > 1$ and $\alpha\left(\frac{\gamma-1}{\beta\gamma e}\right)^{\frac{\gamma-1}{\gamma}} > \frac{C_3}{C_2-C_1}$.

Case A. If $C(0;t) > C(T_2;t)$, then the minimum of $C(T;t)$ is at $T = T_2$ for $t \geq T_2$ and is at $T = t$ for $t < T_2$.

Case B. If $C(0;t) < C(T_2;t)$, then the minimum of $C(T;t)$ is at $T = 0$.

Case C. If $C(0;t) = C(T_2;t)$, then the minimum of $C(T;t)$ is at 0 or T_2 for $t \geq T_2$ and is at $T = 0$ for $t < T_2$.

Case When $\gamma = 1$

For this case, $\Lambda(T) = \alpha e^{-\beta T}$ where $\alpha = ab$ and $\beta = b$. Now $\Lambda(T)$ is a monotonically decreasing function of T and $\Lambda(0) = \alpha$. If $\alpha \leq \frac{c_3}{c_2 - c_1}$, (4.12) has no feasible solution and $\frac{dC(T;t)}{dT} > 0$ for $T \geq 0$. Therefore, the minimum of $C(T;t)$ is at $T = 0$, i.e., $T^* = 0$. If, however, $\alpha > \frac{c_3}{c_2 - c_1}$, then there exists a unique solution of (4.12) given by

$$T_0 = \frac{1}{\beta} \cdot \ell n\{\frac{\alpha(c_2 - c_1)}{c_3}\} \quad . \tag{4.16}$$

From the fact that $\frac{dC(T;t)}{dT} < 0$ for $0 \leq T < T_0$ and $\frac{dC(T;t)}{dT} > 0$ for $T > T_0$, the minimum of $C(T;t)$ is at $T = T_0$ for $t \geq T_0$ and at $T = t$ for $t < T_0$.

Case When $0 < \gamma < 1$

For this case, $\Lambda(T)$ is a monotonically decreasing function of T with $\Lambda(0) = \infty$ and $\Lambda(\infty) = 0$. Then the unique positive root of Equation (4.12) is the solution. Further, since $\frac{dC(T;t)}{dT} < 0$ for $0 \leq T < T_3$ and $\frac{dC(T;t)}{dT} > 0$ for $T > T_3$, the minimum of $C(T;t)$ is at $T = T_3$ for $t \leq T_3$ and at $T = t$ for $t < T_3$ as seen in Figure 4.8. We can summarize the above results in the following theorem.

Theorem 4.2. Suppose that α, β, γ, c_1, $c_2 (> c_1)$, and c_3 are all greater than zero. Then the optimum release time T^* is given by the following expressions for the cases when $\gamma > 1$, $\gamma = 1$, and $0 < \gamma < 1$.

Case When $\gamma > 1$

Case A: If $\alpha(\frac{\gamma-1}{\beta\gamma e})^{(\gamma-1)/\gamma} < \frac{c_3}{c_2 - c_1}$, then $T^* = 0$.

Case B. If $\alpha(\frac{\gamma-1}{\beta\gamma e})^{(\gamma-1)/\gamma} > \frac{c_3}{c_2 - c_1}$, then there exist two feasible solutions $T = T_1$ and $T = T_2$ $(0 < T_1 < T_2 < \infty)$ which are the two positive roots of Equation (4.12) and the optimum release time is as follows:

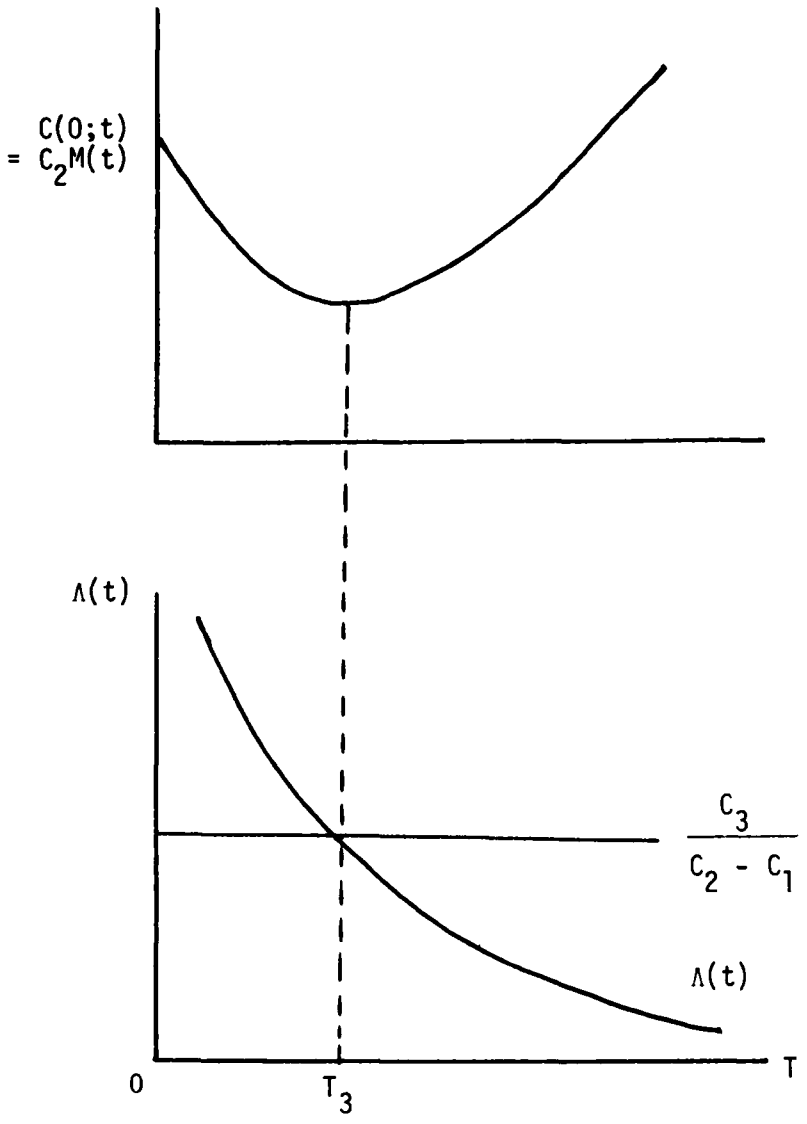


FIG. 4.8. Plots of C(T;t) and Λ(T) for $0 < \gamma < 1$.

If $C(0;t) > C(T_2;t)$, then $T^* = \min\{T_2;t\}$

If $C(0;t) < C(T_2;t)$, then $T^* = 0$

If $C(0;t) = C(T_2;t)$, then $T^* = 0$ and T_2, $t \geq T_2$ and $T^* = 0$ for $t < T_2$

Case When $\gamma = 1$

If $\alpha \leq \dfrac{c_3}{c_2 - c_1}$, then $T^* = 0$. If $\alpha > \dfrac{c_3}{c_2 - c_1}$, then there exists a unique solution

$$T_0 = \frac{1}{\beta} \ell n\{\frac{\alpha(c_2 - c_1)}{c_3}\}$$

of Equation (4.12) and the optimum release time is $T^* = \min(T_0;t)$.

Case When $0 < \gamma < 1$

For this case, Equation (4.12) has a unique positive root T_3 which is the solution, and the optimum release time is $T^* = \min\{T_3;t\}$.

5. BIBLIOGRAPHY

[ABR65] Abramowitz, M. and Stegun, I.A. (1965), Handbook of Mathematical Functions, Dover Publications, Inc.

[ALL78] Allen, A.O. (1978), Probability, Statistics and Queueing Theory, Academic Press.

[AND79] Anderson, T. and Randell, B. (Editors) (1979), Computing System Reliability, An Advanced Course, Cambridge University Press, London.

[ANG80] Angus, J.E., Schafer, R.E., and Sukert, A. (1980), "Software Reliability Model Validation," Proc. Annual Reliability and Maintainability Symposium, San Francisco, CA, pp. 191-193.

[AVI77] Avizienis, A. (1977), "Fault-Tolerant Computing: Progress, Problems, and Prospects," Proc. IFIP Congress 1977, Toronto, Canada, pp. 405-420.

[BAR75] Barlow, R.E. and Proschan, F. (1975), Statistical Theory of Reliability and Life Testing: Probability Models, Holt, Rinehart and Winston, Inc.

[BAS74] Basin, S.L. (1974), Estimation of Software Error Rate Via Capture-Recapture Sampling, Science Applications, Inc., Palo Alto, CA.

[BAS80] Bastani, F.B., "An Input Domain Based Theory of Software Reliability and Its Application," Ph.D. Dissertation, University of California, Berkeley, 1980.

[BEI79] Beightler, C.S., Phillips, D.T., and Wilde, D.J. (1979), Foundations of Optimization, Prentice-Hall.

[BEL76] Belady, L.A. and Lehman, M.M. (1976), "A Model of Large Program Development," IBM Systems Journal, Vol. 15, No. 3, pp. 225-252.

[BOE76] Boehm, B.W., Brown, J.R., and Lipow, M. (1976), "Quantitative Evaluation of Software Quality," Proc. 2nd International Conference on Software Engineering.

[BRO75] Brown, J.R., Lipow, M., "Testing for Software Reliability," Proc. 1975 International Conference Reliable Software, Los Angeles, CA, April, 1975, pp. 518-527.

[BRO72] Brown, M. (1972), "Statistical Analysis of Non-Homogeneous Poisson Processes," in Stochastic Point Processes, edited by P.A.W. Lewis, John Wiley & Sons, New York, pp. 67-89.

[CAS80] Castillo, X. and Siewiorek, D.P. (1980), "A Performance Reliability Model for Computing Systems," Dept. of Computer Science, Carnegie-Mellon University.

[CHA78] Champine, G.A. (1978), "What Makes a System Reliable," Datamation, pp. 195-206.

[CHA78] Chandy, K.M. and Yeh, R.T. (Editors) (1978), Current Trends in Programming Methodology, Vol. III: Software Modeling, Prentice-Hall.

[CHO80] Cho, C.K., (1980), An Introduction to Software Quality Control, John Wiley & Sons.

[CIN75] Cinlar, E. (1975), Introduction to Stochastic Processes, Prentice-Hall.

[COX65] Cox, D.R. and Miller, H.D.., (1965), The Theory of Stochastic Processes, Wiley and Sons, Inc.

[COX66] Cox, D.R. and Lewis, P.A.W., (1966), The Statistical Analysis of Series of Events, Methuen, London.

[DAL77] Daly, E.B. (1977), "Management of Software Development," IEEE Trans. on Software Engineering, pp. 230-242.

[DON75] Donelson, J., III, (1975), Duane's Reliability Growth Model as a Non-Homogeneous Poisson Process, IDA Log. No. HQ76-18012, Paper P-1162.

[DUA64] Duane, J.T. (1964), "Learning Curve Approach to Reliability Monitoring," IEEE Trans. Aerospace, Vol. 2, pp. 563-566.

[DUN82] Dunn, R. and Ullman, R. (1982), Quality Assurance for Computer Software, McGraw-Hill, Book Co.

[DUR80] Duran, J.W. and Wiorkowski, J.J. (1980), "Quantifying Software Validity by Sampling," IEEE Trans. on Reliability, Vol. R-29, No. 2.

[END75] Endres, A., "An Analysis of Errors and Their Causes in System Programs," Proc. International Conference on Reliability Software, Los Angeles, CA, pp. 327-336.

[FEL57] Feller, W. (1957), An Introduction to Probability Theory and Its Applications, 2nd Ed., Vol. I, John Wiley & Sons, Inc.

[FEL66] Feller, W. (1966), An Introduction to Probability Theory and Its Applications, Vol. II, Wiley.

[FIN76] Finkelstein, J.M. (1976), "Confidence Bounds on the Parameters of the Weibull Process," Technometrics, Vol. 18, No. 1, pp. 115-117.

[FOR77] Forman, E.H. and Singpurwalla, N.D. (1977), "An Empirical Stopping Rule for Debugging and Testing Computer Software," Journal of the American Statistical Association, Vol. 72, No. 360, pp. 750-757.

[FRI77] Fries, M.J. (1977), Software Error Data Acquisition, Boeing Aerospace Co., Final Technical Report, RADC-TR-77-130, AD A039-916.

[GER76] Gerhart, S. and Yelowitz, L. (1976), "Observations of Fallibility in Applications of Modern Programming Methodologies," IEEE Transactions on Software Engineering.

[GIR73] Girard, E. and Rault, J.C. (1973), "A Programming Technique for Software Reliability," IEEE Symposium on Computer Software Reliability.

[GLA79] Glass, R.L. (1979), Software Reliability Guidebook, Prentice-Hall, Inc.

[GLA81] Glass, R.L. (1981), "Persistent Software Errors," IEEE Transactions on Software Engineering, Vol. SE-7.

[GOE77] Goel, A.L. (1977), Summary of Technical Progress: Bayesian Software Reliability Prediction Models, RADC-TR-77-112, Syracuse University, AD A039-022.

[GOE78] Goel, A.L. and Okumoto, K. (1978), Bayesian Software Correction Limit Policies, Final Technical Report, Syracuse University, RADC-TR-78-155, Vol. 2 (of 5), AD A057-872.

[GOE78] Goel, A.L. and Okumoto, K. (1978), An Imperfect Debugging Model for Software Reliability, Final Technical Report, Syracuse University, RADC-TR-78-155, Vol. 1 (of 5), AD A057-879.

[GOE78] Goel, A.L. and Okumoto, K. (1978), "A Time Dependent Error Detection Rate Model for a Large Scale Software System," Proc. Third USA-Japan Computer Conference, San Francisco, CA, pp. 35-40.

[GOE78] Goel, A.L. and Okumoto, K. (1978), "An Analysis of Recurrent Software Failures in a Real-Time Control System," Proc. Annual Technical Conference, ACM, Washington, DC, pp. 496-500.

[GOE79] Goel, A.L. (1979), "Reliability and Other Performance Measures of Computer Software," Proc. First International Conference on Reliability and Exploitation of Computer Systems, Wroclaw, Poland, pp. 23-31.

[GOE79] Goel, A.L. and Okumoto, K. (1979), "A Time Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures," IEEE Transactions on Reliability, Vol. R-28, No. 3, pp. 206-211.

[GOE79] Goel, A.L. and Okumoto, K. (1979), "A Markovian Model for Reliability and Other Performance Measures of Software Systems," Proc. National Computer Conference, New York, Vol. 48, pp. 769-774.

[GOE80] Goel, A.L. (1980), "A Software Error Detection Model with Applications," Journal of Systems and Software (to appear).

[GOE80] Goel, A.L. (1980), "A Summary of the Discussion On An Analysis of Computer Software Reliability Models," IEEE Transactions on Software Engineering, Vol. SE-6, No. 5, pp. 501-502.

[GOE82] Goel, A.L. (1982), "Software Reliability Modelling: An Overview and a Case Study," International Journal of Reliability and Safety (to appear).

[GOO77] Goodenough, J. and Gerhart, S. (1977), "Toward A Theory of Testing: Data Selection Criteria," Current Trends in Programming Methodology, Vol. 2, R.T. Yeh, Ed., Englewood Cliffs, NJ, Prentice-Hall.

[GOO79] Goodenough, J. (1979), "A Survey of Program Testing Issues," Research Directions in Software Technology, The MIT Press.

[GOO80] Goodenough, J. (1980), "The ADA Compiler Validation Capability," Proc. ACM SIGPLAN Symp. on the ADA Programming Language, Boston.

[GRI78] Griffiths, S.N. (1978), "Design Methodologies -- A Comparison," Structured Analysis and Design, Vol. II, Infotech Inter-Limited.

[HAL77] Halsted, M.H. (1977), Elements of Software Science, Elsevier North Holland Publishing Co., New York.

[HAN76] Han, Y. (1976), "A Systematic Study of Computer System Reliability," Ph.D. Thesis, University of California, Berkeley.

[HO78] Ho, Siu-Bun Franklin, "A Systematic Approach to the Development and Validation of Software for Critical Applications," Ph.D. Dissertation, University of California, Berkeley, 1978.

[HOW80] Howden, W. (1980), "Functional Program Testing," IEEE Transactions on Software Engineering, Vol. SE-6, No. 2.

[HUA75] Huang, J.C. (1975), "An Approach to Program Testing," Computing Surveys, Vol. 7, No. 3.

[JEL72] Jelinski, Z. and Moranda, P. (1972), "Software Reliability Research," in Statistical Computer Performance Evaluation, W. Freiberger (Ed.), Academic Press, pp. 465-484.

[JEN79] Jensen, R.W. and Tonies, C.C. (1979), Software Engineering, Prentice-Hall, Inc.

[JON80] Jones, C.B. (1980), Software Development - A Rigorous Approach, Prentice-Hall International Series in Computer Science, Longon.

[KAR75] Karlin, S. and Taylor, H.M. (1975), A First Course in Stochastic Processes, Academic Press, New York.

[KLE76] Kleinrock, L. (1976), Queueing Systems, Vol. II: Computer Applications, John Wiley & Sons, Inc., New York.

[KOB78] Kobayashi, H. (1978), Modeling and Analysis: An Introduction to System Performance Evaluation Methodology, Addison-Wesley.

[LAN77] Landrault, C. and Laprie, J.S. (1977), "Reliability and Availability Modeling of Systems Featuring Hardware and Software Faults," Proc. 7th Annual International Conference on Fault-Tolerant Computing, Los Angeles, CA.

[LEW64] Lewis, P.A.W. (1964), "Implications of a Failure Model for the Use and Maintenance of Computers," Journal of Applied Probability, Vol. 1, pp. 347-368.

[LEW76] Lewis, P.A.W.and Shedler, G.S. (1976), "Statistical Analysis of Non-stationary Series of Events in a Data Base System," IBM Journal of Research and Development, Vol. 20, pp. 465-482.

[LIP72] Lipow, M. (1972), Estimation of Software Package Residual Errors, TRW Software Series Report, TRW-SS-72-09, Redondo Beach, CA.

[LIP73] Lipow, M. (1973), Maximum Likelihood Estimation of Parameters of a Software Time-to-Failure Distribution, TRW Systems Group Report, 2260.1.9-73B-15, Redondo Beach, CA.

[LIT73] Littlewood, B. and Verall, J.L. (1973), "A Bayesian Reliability Growth Model for Computer Software," Applied Statistics, Vol. 22, No. 3, pp. 332-346.

[LIT75] Littlewood, B. (1975), "A Reliability Model for Systems with Markov Structure," Applied Statistics, Vol. 24, No. 2, pp. 172-177.

[LIT76] Littlewood, B. (1976), "A Semi-Markov Model for Software Reliability with Failure Costs," Proc. MRI Symposium on Software Engineering, New York, pp. 281-300.

[LIT80] Littlewood, B. (1980), "Theories of Software Reliability: How Good Are They and How Can They Be Improved?" IEEE Transactions on Software Engineering, Vol. SE-6, No. 5.

[MAG52] Maguire, B.A., Pearson, E.S., and Wynn, A.H.A. (1952), "The Time Intervals Between Industrial Accidents," Biometrika, Vol. 39, pp. 168-180.

[MAN74] Mann, N.R., Schafer, R.E., and Singpurwalla, N.D. (1974), Methods for Statistical Analysis of Reliability and Life Data, John Wiley & Sons.

[MEY78] Meyer, J.F. (1978), "On Evaluating the Performability of Degradable Computing Systems," Proc. 8th Annual International Conference on Fault-Tolerant Computing, pp. 44-49.

[MIL76] Miller, D.R. (1976), "Order Statistics, Poisson Processes and Repairable Systems," Journal of Applied Probability, Vol. 13, pp. 519-529.

[MIL72] Mills, H.D. (1972), On the Statistical Validation of Computer Programs, IBM Federal Systems Division, Gaithersburg, MD, Report 72-6015.

[MIY75] Miyamoto, I. (1975), "Software Reliability in On-Line Real Time Environment," Proc. International Conference on Reliable Software, Los Angeles, CA, pp. 194-203.

[MOE76] Moeller, S.K. (1976), "The Rasch-Weibull Process," Scandanavian Journal of Statistics, Vol. 3, pp. 107-115.

[MOR75] Moranda, P.B. (1975), "Prediction of Software Reliability During Debugging," Proc. Annual Reliability and Maintainability Symposium, Washington, DC, pp. 327-332.

[MOR75] Moranda, P.B. (1975), "A Comparison of Software Error-Rate Models," Proc. 1975 Texas Conference on Computing, pp. 2A6.1-6.9.

[MUS75] Musa, J.D. (1975), "A Theory of Software Reliability and Its Application," IEEE Trans. on Software Engineering, Vol. SE-1, No. 3, pp. 312-327.

[MUS80] Musa, J.D., Software Reliability Data, DACS, RADC, New York.

[MUT77] Muth, E.J. (1977), Transform Method with Applications to Engineering and Operations Research, Prentice-Hall.

[MYE75] Myers, G.J. (1975), Reliable Software Through Composite Design, Petrocelli/Charter, New York.

[MYE76] Myers, G.J. (1976), Software Reliability, Principles and Practices, John Wiley & Sons, New York.

[MYE79] Myers, G.J. (1979), The Art of Software Testing, John Wiley & Sons.

[NEL75] Nelson, E.C. (1975), Software Reliability, TRW Software Series, TRW-SS-75-05, Redondo Beach, CA.

[OKU78] Okumoto, K. and Goel, A.L. (1978), Classical and Bayesian Inference for the Software Imperfect Debugging Model, Syracuse University, Final Technical Report, RADC-TR-78-155, Vol. 2 (of 5), AD A057-871.

[OKU78] Okumoto, K. and Goel, A.L. (1978), Availability Analysis of Software Systems Under Imperfect Maintenance, Syracuse University, Final Technical Report, RADC-TR-78-155, Vol. 3 (of 5), AD A057-872.

[OKU78] Okumoto, K. and Goel, A.L. (1978), "Availability and Other Performance Measures of Software Systems Under Imperfect Maintenance," Proc. COMPSAC, 1978, pp. 66-71.

[ORR77] Orr, K.T. (1977), "Using Structured Systems Design," Structured Systems Development, Yourdon Press.

[ORR78] Orr, K.T. (1978), "Introducing Structured Systems Design," Structured Analysis and Design, Vol. II, Infotech International Limited.

[PIE76] Pierskalla, W.P. and Voelker, J.A. (1976), "A Survey of Maintenance Models: The Control and Surveillance of Deteriorating Systems," Naval Research Logistics Quarterly, Vol. 23, No. 3, pp. 353-388.

[PRO63] Proschan, F. (1963), "Theoretical Explanation of Observed Decreasing Failure Rate," Technometrics, Vol. 5, No. 3, pp. 375-383.

[PYK61] Pyke, R. (1961), "Markov Renewal Processes: Definitions and Preliminary Properties," Annals of Mathematical Statistics, Vol. 32, pp. 1231-1242.

[RAH78] Raha, D. and Silva, N. (1978), "Digital Communication Systems - Reliability Trends," Proc. 1979 Annual Reliability and Maintenance Symposium, pp. 452-459.

[ROH76] Rohatgi, V.K. (1976), An Introduction to Probability Theory and Mathematical Statistics, John Wiley & Sons.

[ROS76] Ross, S.M. (1976), Applied Probability Models with Optimization Applications, Holden-Day.

[ROU73] Roussas, G.G. (1973), A First Course in Mathematical Statistics, Addison Wesley Publishing Co.

[RYE77] Rye, P. et al. (1977), Software Systems Development: A CSDL Project History, The Charles Start Draper Laboratory, Inc., Final Technical Report, RADC-TR-77-213, AD A042-186.

[SCH79] Schafer, R.E., et al. (1979), Validation of Software Reliability Models, Huges Aircraft Co., Final Technical Report, RADC-TR-78-147, AD-A072-113.

[SCH73] Schick, G.J. and Wolverton, R.W. (1973), "Assessment of Software Reliability," 11th Annual Meeting of the German Operations Research Society, DGOR, Hamburg, Germany; also in Proc. Operations Research, Physica-Verlag, Wurzberg-Wien, pp. 395-422.

[SCH78] Schick, G.J. and Wolverton, R.W. (1978), "An Analysis of Computing Software Reliability Models," IEEE Trans. on Software Engineering, Vol. SE-4, No. 2, pp. 104-120.

[SCH72] Schneidewind, N.F. (1972), "An Approach to Software Reliability Prediction and Quality Control," Proc. AFIPS Fall Joint Computer Conference, Vol. 41, Part II, pp. 837-838.

[SCH75] Schneidewind, N.F. (1975), "Analysis of Error Processes in Computer Software," Proc. International Conference on Reliable Software, Los Angeles, CA, pp. 337-346.

[SHO72] Shooman, M.L. (1972), "Probabilistic Models for Software Reliability Prediction," Statistical Computer Performance Evaluation, W. Freiberger (Ed.), Academic Press, pp. 485-502.

[SHO75] Shooman, M.L. (1975), "Software Reliability Measurement and Models," Proc. 1975 Annual Reliability and Maintainability Symposium, pp. 485-491.

[SHO76] Shooman, M.L. (1976), "Structural Models for Software Reliability and Prediction," Proc. 2nd International Conference on Software Engineering, pp. 268-273.

[SNY75] Snyder, D.L. (1975), Random Point Processes, John Wiley & Sons.

[SUK76] Sukert, A.N. (1976), A Software Reliability Modelling Study, In-house Technical Report, RADC-TR-76-247, AD A030-437.

[SUK77] Sukert, A.N. (1977), "An Investigation of Software Reliability Models," Proc. Annual Reliability and Maintainability Symposium, Philadelphia, PA, pp. 478-484.

[SUK78] Sukert, A.N. and Goel, A.L. (1978), "Error Modelling Applications in Software Quality Assurance," Proc. Software Quality and Assurance Workshop, San Diego, CA, pp. 33-38.

[SUK80] Sukert, A.N. and Goel, A.L. (1980), "A Guidebook for Software Reliability Assessment," Proc. Annual Reliability and Maintainability Symposium, San Francisco, CA, pp. 188-190.

[TAI80] Tai, K.C. (1980), "Program Testing Complexity and Test Criteria," IEEE Transactions on Software Engineering, Vol. SE-6, No. 6.

[THA76] Thayer, T.A., Lipow, M., and Nelson, E.C. (1976), Software Reliability Study, TRW Defense and Space Systems Group, Final Technical Report RADC-TR-76-238, AD A030-798.

[THA78] Thayer, T.A., Lipow, M., and Nelson, E.C. (1978), Software Reliability, North-Holland, Amsterdam.

[TRI75] Trivedi, A.K. (1975), "Computer Software Reliability: Many-State Markov Modelling Techniques," Ph.D. Dissertation, Dept. of Electrical Engineering, Polytechnic Institute of New York, NY.

[TRI74] Trivedi, A.K. and Shooman, M.L. (1974), "A Markov Model for the Evaluation of Computer Software Performance," Research Report, Polytechnic Institute EE/EP, 74-011-EER 110.

[TRI75] Trivedi, A.K. and Shooman, M.L. (1975), Computer Software Reliability: Many State Markov Modelling Techniques, Polytechnic Institute of New York, Interim Report, RADC-TR-75-169, AD 014-824.

[WAG73] Wagoner, W.L. (1973), The Final Report on A Software Reliability Measurement Study, Technology Division, The Aerospace Corp., Report No. TOR-0074 (4112)-1, El Segundo, CA.

[WIL77] Willman, H.E., Jr., et al. (1977), Software Systems Reliability: A Raytheon Project History, Raytheon Co., Final Technical Report, RADC-TR-77-188.

[WUL75] Wulf, W.A. (1975), "Reliable Hardware/Software Architecture," IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, pp. 233-240.

[YAU79] Yau, S.S. and MacGregor, T.E. (1979), On Software Reliability Modeling, Interim Report, Northwestern University, RADC-TR-79-129.

[YOU72] Yourdon, E. (1972), "Reliability Measurements for Third Generation Computer Systems," Proc. Annual Reliability and Maintainability Symposium, pp. 174-183.

[YOU75] Yourdon, E. (1975), Techniques of Program Structure and Design, Prentice-Hall, Inc., Englewood Cliffs, NJ.

A1. INTRODUCTION

In this section we describe the development of stochastic models for performance and cost evaluation of hardware-software systems in the operational phase.

Section A.2 deals with the development of stochastic models for system performance assessment. The state of the system is described by up or down states of the hardware and the software system and by the number of errors in the software system. The hardware-software system is down if either the hardware or the software system is down, and up if both are up. The hardware failure distribution is exponential with failure rate β. The software failure distribution between occurrences of software failures is also exponential with a failure rate $i\lambda$, where $i = 0,1,\ldots,N$ is the number of remaining errors in the system. The repair rates are exponential with parameters γ and μ, and the probabilities of imperfect repair are p_h and p_s for the hardware and the software systems, respectively.

Based on this model, expressions for various stochastic performance measures are also developed in Section A2. These are distribution of time to a specified number of remaining software errors; state

occupancy probabilities; expected number of hardware, software, and hardware-software failures detected by time t; system reliability, availability and average availability.

In some cases, an improvement in one performance measure causes a worsening of another. For example, an improved system availability causes an increase in the expected number of failures. In order to evaluate the effect of these conflicting measures on system performance, cost models are developed in Section A3 for the hardware, software, and hardware-software systems. Each model gives expected total cost by time t and consists of three cost elements; the cost of failures, the cost of repairs, and the cost due to system unavailability. The results of a numerical study to investigate the effects of cost factors, failure rates, and repair rates on the expected number of failures, average availability and expected total cost/unit time are also discussed.

A2. A MARKOV MODEL FOR HARDWARE-SOFTWARE SYSTEM AND PERFORMANCE MEASURES

In this section we develop a stochastic model and expressions for the performance measures of a hardware-software system. The basic model is developed in Section A2.1 and assumes the system behavior to be Markovian.

In order to use this model to evaluate and predict the system performance, we generally need expressions for the appropriate quantitative measures. Such expressions for the following measures are derived in Sections A2.2 to A2.5.

(i) Distribution of time to a specified number of remaining errors in the software system.

(ii) State occupancy probabilities.

(iii) System reliability and availability.

(iv) Expected number of software, hardware, and total failures by time t.

A2.1. System Description and Model Development

Consider a system consisting of hardware and software components, all of which are subject to random failures. The hardware components fail due to either defects or wear-out.

A software component is said to fail when a fault, a specific manifestation of an error in the program, is evoked by some input data resulting in the program not correctly computing the required function. Whenever any of these failures occurs, the system goes out of operation. A repair activity is then undertaken to remove the cause of the failure and bring the system back to an operational state.

In the present study, we assume that the hardware and software components can be viewed as a single system each. In other words, the hardware-software system will be treated as 2-unit (or 2-system) systems, one representing the hardware components and the other the software components. The up and down states of such a system are shown in Figure A2.1.

We develop a model for the stochastic behavior of the system under the following assumptions:

(i) The errors in the software system are independent of each other and each has an error occurrence rate λ.

(ii) The failures of the hardware system are independent of each other and have a constant occurrence rate β. Only those failures which cause the system to go down are considered.

(iii) The probability of two or more software or hardware failures occurring simultaneously

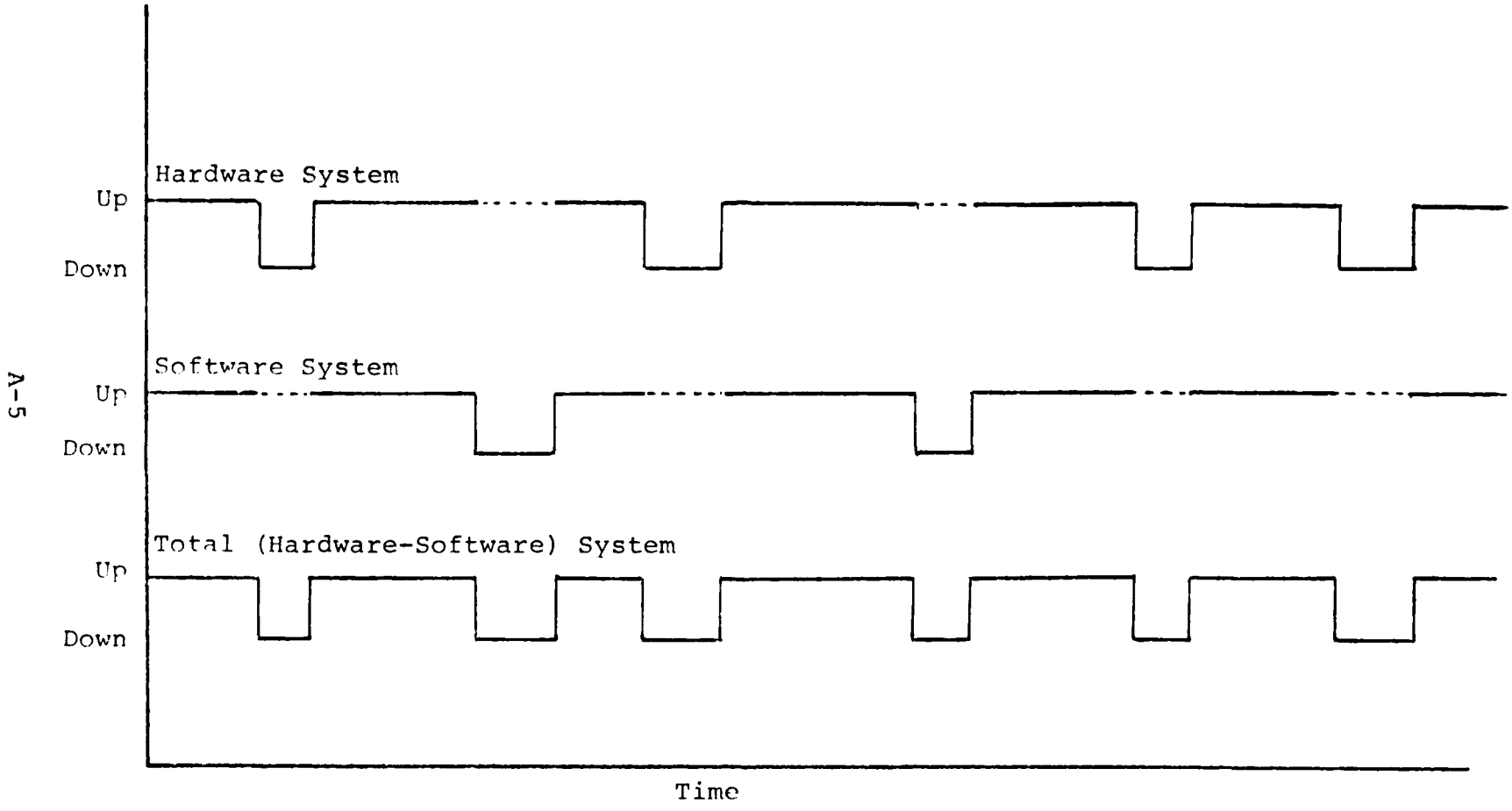


Figure A2.1. Up and Down Behavior of Hardware-Software System.

is negligible.

(iv) The time to remove a software error, when there are i errors in the system, follows an exponential distribution with parameter μ_i.

(v) The time to remove the cause of a hardware failure follows an exponential distribution with parameter γ.

(vi) Failures and repairs of the hardware system are independent of both the failures and repairs of the software system.

(vii) At most one software error is removed at correction time and no new software errors are introduced during the error removal (correction) phase.

(viii) When the system is inoperative due to the occurrence of a software failure, the error causing the failure, when detected, is corrected with probability p_s $(0 \leq p_s \leq 1)$, while with probability $q_s (p_s + q_s = 1)$ the error is not removed. Thus, q_s is the probability of imperfect maintenance of software.

(ix) After the occurrence of a hardware failure, the cause of the failure is removed with probability $p_h (0 \leq p_h \leq 1)$ while with probability $q_h (p_h + q_h = 1)$, the cause is not removed. Thus, q_h is

the probability of imperfect maintenance of hardware.

(x) The system is considered to be inoperative whenever it is under maintenance following a hardware or a software failure.

Now, we examine the failure and repair times of the software and hardware systems independently, based upon the above assumptions.

Software failures, from assumptions (i) and (iii), follow an exponential distribution. Let i be the number of errors in the software system. Then the probability density function (pdf) of the time to next software failure, T_i, is given by the distribution of the first order statistic of i exponential distributions each with parameter λ, i.e.

$$f_i(t) = \binom{i}{1}(\lambda e^{-\lambda t})(e^{-\lambda t})^{i-1}$$

or

$$f_i(t) = i\lambda \cdot e^{-i\lambda t} \qquad \text{(A2.1)}$$

Letting $\lambda_i = i\lambda$, the pdf and the cumulative distribution function (cdf) of T_i can be written as

$$f_i(t) = \lambda_i e^{-\lambda_i t} \qquad \text{(A2.2)}$$

and

$$F_i(t) = 1 - e^{-\lambda_i t} \qquad \text{(A2.3)}$$

From assumption (iv), the cdf of the software maintenance

time when there are i errors in the system, W_i, is

$$P(W_i \leq t) = 1 - e^{-\mu_i t} \tag{A2.4}$$

The cdf's of the hardware time to failure, U, and maintenance time, V, from assumptions (ii) and (v), respectively, are:

$$P(U \leq t) = 1 - e^{-\beta t} \tag{A2.5}$$

and

$$P(V \leq t) = 1 - e^{-\gamma t}. \tag{A2.6}$$

To summarize, hardware failures and repairs occur according to exponential distributions with parameters β and γ, respectively. These parameters are considered to remain constant. The distribution of the times between software failures also follows an exponential distribution, but its parameter, λ_i also varies with the number of errors remaining in the software system, i. The distribution of the maintenance time for software is again exponential with a parameter μ_i which changes with i.

Now, we consider the failure phenomenon in the total hardware-software system. Suppose there are i errors in the software and the total system is operational. Let

$$Y_i = \min(T_i, U) \tag{A2.7}$$

It can be easily shown that Y_i has an exponential distribution with parameter $(\beta+\lambda_i)$ and

$$F_{Y_i}(y) = 1 - e^{-(\beta+\lambda_i)y} \tag{A2.8}$$

The probability that a software failure will occur before a hardware failure is

$$P(T_i<U) = \int_0^\infty P(U>T_i \mid T_i=t) \cdot dF_i(t)$$

$$= \int_0^\infty P(U>t) \cdot \lambda_i e^{-\lambda_i t} \cdot dt$$

$$= \int_0^\infty \lambda_i e^{-(\beta+\lambda_i)t} dt$$

or

$$P(T_i<U) \equiv p_i = \frac{\lambda_i}{\beta + \lambda_i}, \quad i = 0, 1, \ldots, N \tag{A2.9}$$

Similarly, the probability that a hardware failure occurs before a software failure is

$$P(U<T_i) \equiv q_i = \frac{\beta}{\beta + \lambda_i}, \quad i = 0,1, \ldots, N \tag{A2.10}$$

In other words, when the hardware-software system is operational with i software errors, the time to next failure is given by Y_i. The probability of the next failure being a software failure is p_i and being a hardware failure is q_i.

Let X(t) denote the state of the system at time t; where

$$X(t) = \begin{cases} i, & \text{the system is operational while there are i errors remaining in the software system, } i = 0, 1, 2, \ldots, N. \\ i_s, & \text{the system is down for maintenance of software with i software errors, } i_s = 1_s, 2_s, \ldots, N_s \\ i_h, & \text{the system is down for maintenance of hardware with i software errors, } i_h = 0_h, 1_h, \ldots, N_h \end{cases} \tag{A2.11}$$

The transitions between the states of the system, i.e., in X(t), are governed by assumptions (i) through (x) and Equations (A2.9) and (A2.10). The transition probability matrix for the X(t) process is given in Equation (A2.12) and a diagrammatic representation of such transitions is given in Figure A2.2.

$$
P_{k,j} = \begin{array}{c} \\ N \\ N_h \\ N_s \\ N\text{-}1 \\ N\text{-}1_h \\ N\text{-}1_s \\ N\text{-}2 \\ \vdots \\ 1 \\ 1_h \\ 1_s \\ 0 \\ 0_h \end{array}
\begin{array}{c}
\begin{array}{ccccccccccccc} N & N_h & N_s & N\text{-}1 & N\text{-}1_h & N\text{-}1_s & N\text{-}2 & \cdots & 1 & 1_h & 1_s & 0 & 0_h \end{array} \\
\begin{bmatrix}
0 & q_N & p_N & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
p_h & q_h & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
q_s & 0 & 0 & p_s & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & q_{N-1} & p_{N-1} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & p_h & q_h & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & q_s & 0 & 0 & p_s & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & q_1 & p_1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & p_h & q_h & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & q_s & 0 & 0 & p_s & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & p_h & q_h
\end{bmatrix}
\end{array}
\qquad \text{(A2.12)}
$$

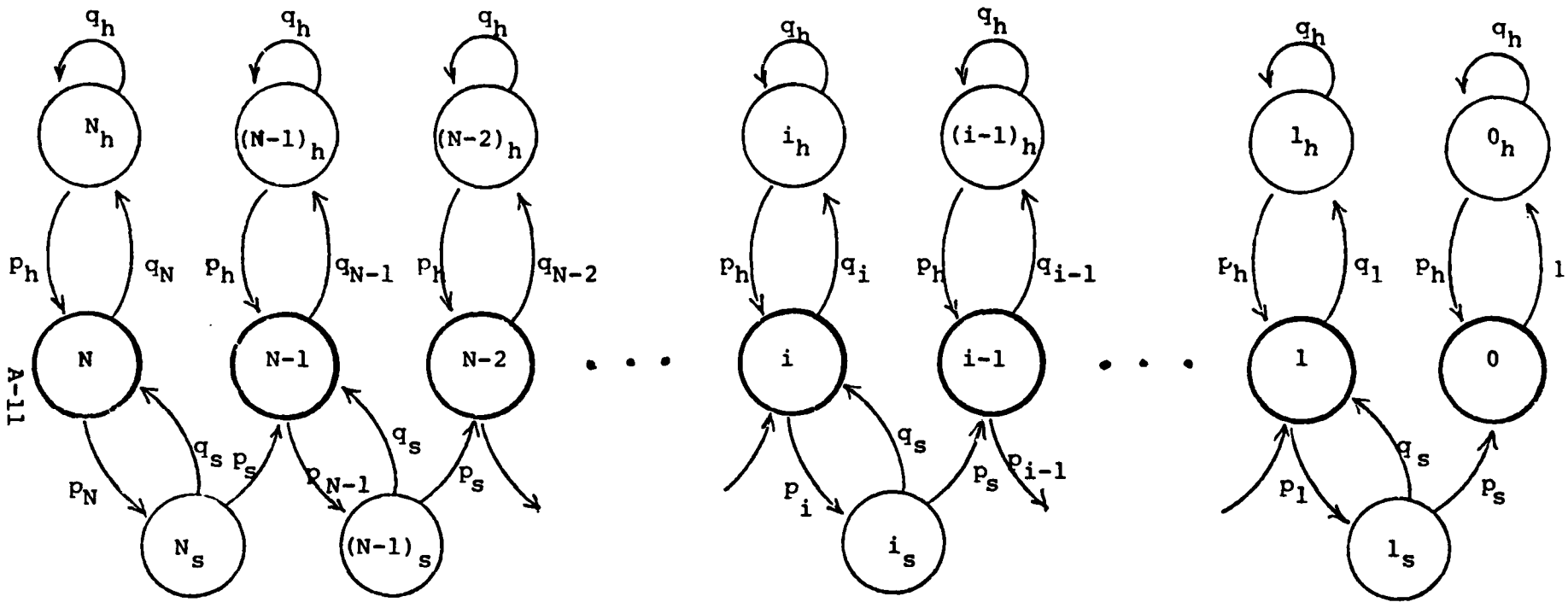


Figure A2.2. Diagrammatic Representation of Transitions Between States of X(t).

To summarize the system behavior, consider once again the above situation, i.e., the total system is operational with i software errors. The time to a failure is governed by Y_i. If a software failure occurs, and the probability of this occurring is p_i, the system goes into a down state i_s. The system undergoes software maintenance and, after a random time governed by Equation (A2.4), goes to state i with probability q_s and to state (i-1) with probability p_s.

If the failure is a hardware failure, and the probability of this happening is q_i, the system goes into a down state, i_h. Following a repair for the failure according to Equation (A2.6), the system goes back to state i with probability p_h or stays in state i_h with probability q_h.

The above system behavior is valid only until the software is error-free. After the software is error-free, the total system reduces to a hardware system only.

Thus, we see that the stochastic process X(t) forms a semi-Markov process. It makes transitions as described above and the times spent in various states are random, given by Y_i, W_i, or V, depending on the state. A typical realization of the X(t) process corresponding to Figure A2.2 is shown in Figure A2.3.

Let $Q_{k,j}(t)$, $k,j = i,i_s,i_h$, be the one step transition probability that after making a transition into state k, the process X(t) next makes a transition into state j in an amount of time less than or equal to t. Then, $Q_{k,j}(t)$

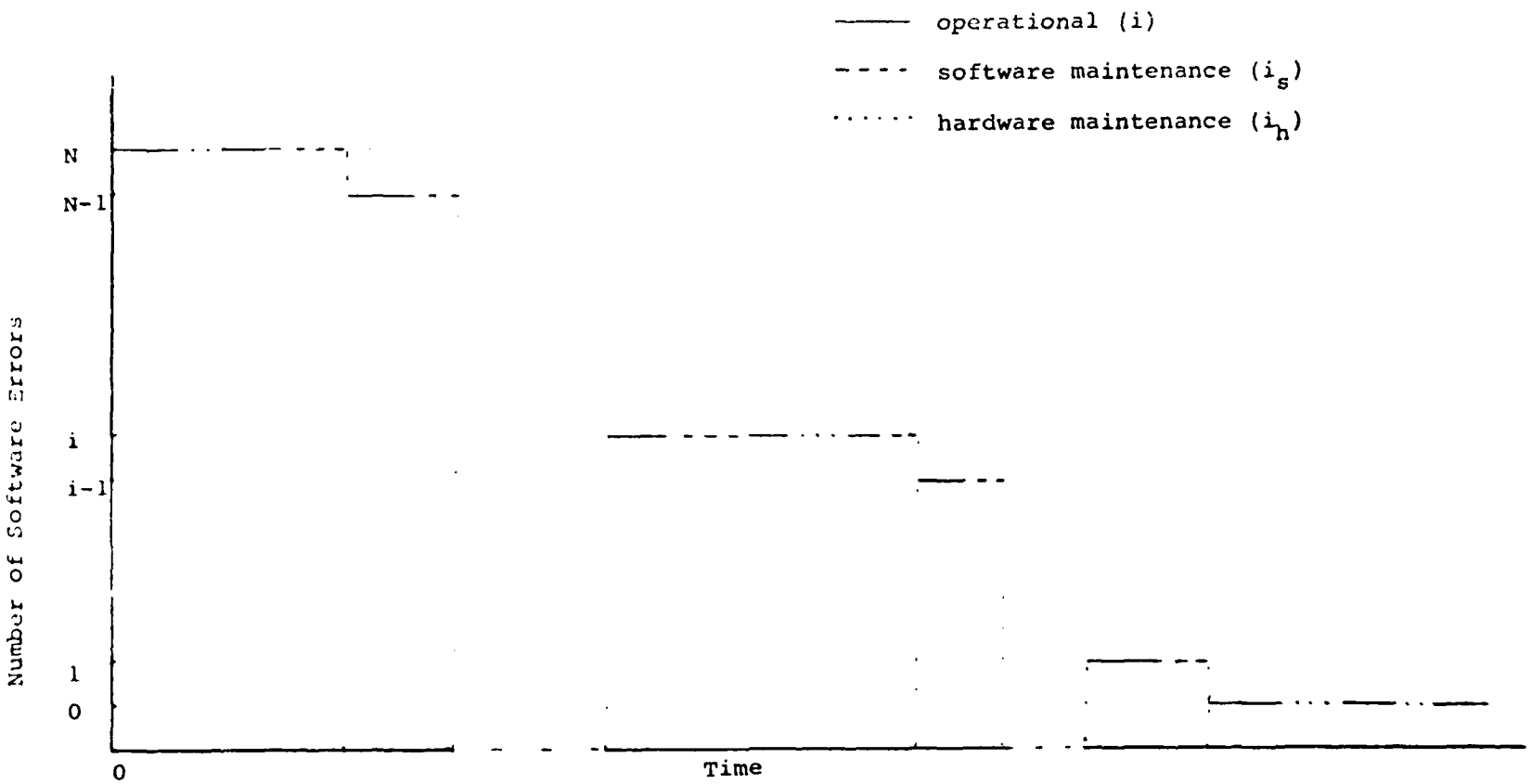


Figure A2.3. A Typical Realization of X(t) Process.

is given by the product of $P_{k,j}$ and the cdf upto t of the time corresponding to state k. Thus, for k = i, and $j = i_s$, we have

$$Q_{i,i_s}(t) = P_{i,i_s} \cdot F_{Y_i}(t) \tag{A2.13}$$

The expressions for various $Q_{k,j}$'s are as follows:

$$\left.\begin{aligned}
Q_{i,i_s}(t) &= \frac{\lambda_i}{\beta + \lambda_i}\left(1 - e^{-(\beta+\lambda_i)t}\right) \\
Q_{i,i_h}(t) &= \frac{\beta}{\beta + \lambda_i}\left(1 - e^{-(\beta+\lambda_i)t}\right) \\
Q_{i_s,i}(t) &= q_s\left(1 - e^{-\mu_i t}\right), \\
Q_{i_s,i-1}(t) &= p_s\left(1 - e^{-\mu_i t}\right), \\
Q_{i_h,i}(t) &= p_h\left(1 - e^{-\gamma t}\right), \\
\text{and} \\
Q_{i_h,i_h}(t) &= q_h\left(1 - e^{-\gamma t}\right).
\end{aligned}\right\} \tag{A2.14}$$

The expressions for $Q_{k,j}(t)$'s given by Equation (A2.14) constitute the basic equations that describe the stochastic behavior of the X(t) process. These equations will be used in the subsequent sections to derive the system performance measures. We will need the Laplace-Stieltjes transforms of the $Q_{k,j}(t)$'s and some related results. These are given below.

Let $\mathcal{L}$ and $\mathcal{L}S$ denote the Laplace and Laplace Stieltjes transform, respectively; and for any function g and G, let

$$g^*(s) = \mathcal{L}(g(t)), \quad \text{and } \tilde{G}(s) = \mathcal{L}S(G(t)).$$

The Laplace-Stieltjes transform of the above $Q_{i,j}(t)$'s are

$$\mathcal{L}S(Q_{i,i_s}) = \frac{\lambda_i}{s + \beta + \lambda_i}, \tag{A2.16}$$

$$\mathcal{L}S(Q_{i,i_h}) = \frac{\beta}{s + \beta + \lambda_i}, \tag{A2.17}$$

$$\mathcal{L}S(Q_{i_s,i}) = \frac{q_s \mu_i}{s + \mu_i}, \tag{A2.18}$$

$$\mathcal{L}S(Q_{i_s,i-1}) = \frac{p_s \mu_i}{s + \mu_i}, \tag{A2.19}$$

$$\mathcal{L}S(Q_{i_h,i}) = \frac{p_h \gamma}{s + \gamma}, \tag{A2.20}$$

$$\mathcal{L}S(Q_{i_h,i_h}) = \frac{q_h \gamma}{s + \gamma}. \tag{A2.21}$$

The following Lemmas from the basic Laplace, Laplace-Stieltjes transforms and their inverses will be useful for our analysis (see Abramowitz et al., 1965, and Muth, 1977).

Lemma A2.1. (Linearity property). If

$$h(t) = Af(t) + Bg(t)$$

then

$$h^*(s) = \mathcal{L}(h(t)) = Af^*(s) + Bg^*(s).$$

Lemma A2.2. The Laplace transform of pdf f(t) is equivalent to the Laplace-Stieltjes transform of its cdf F(t).

$$f^*(s) = \mathcal{L}\{f(t)\} = \int_0^\infty e^{-st} f(t)dt$$

$$= \int_0^\infty e^{-st} dF(t)$$

$$= \mathcal{L}S\{F(t)\}.$$

Lemma A2.3. (Heaviside Expansion Theorem). If

(i) $q(s) = (s-a_1)(s-a_2) \ldots (s-a_m)$,

where $a_1 \neq a_2 \neq \ldots \neq a_m$,

(ii) $p(s)$ is a polynomial of degree m, and

(iii) $f^*(s) = \frac{p(s)}{q(s)}$,

then

$$f(t) = \sum_{n=1}^{m} \frac{p(a_n)}{q'(a_n)} e^{a_n t},$$

where $q'(a_i) = \prod_{\substack{i,j=1 \\ i \neq j}}^{m} (a_i - a_j)$,

and

$$F(t) = \sum_{n=1}^{m} \frac{p(a_n)}{q'(a_n)} \cdot \frac{1}{a_n} (e^{a_n t} - 1).$$

If one of the a_n, i.e. $a_i = 0$, $1 \leq i \leq m$, then

$$F(t) = \frac{p(a_i)}{q'(a_i)} t + \sum_{\substack{n=1 \\ n \neq i}} \frac{p(a_n)}{q'(a_n)} \frac{1}{a_n} (e^{a_n t} - 1).$$

A2.2 Distribution of Time to a Specified Number of Remaining Errors in a Software System

The errors remaining in the software system are sources of failures and we would like to remove them as soon as possible. However, it is not always feasible and/or practical to remove all of them in a reasonable time. In that case, we would like to know the distribution of time to n ($0 \leq n \leq N$) remaining errors.

Let $T_{i,n}$ be the first passage time for operational state i to operational state n and let $G_{i,n}$ be its cdf. Now we derive the equations for $g_{N,n}(t)$ and $G_{N,n}(t)$, the pdf and cdf, respectively, of $T_{N,n}$.

A2.2.1 Distribution of $T_{N,n}$

Consider a time interval (r,r+dr). For any i, the probability of going from i to i_s in this interval is $dQ_{i,i_s}(r)$ and the probability of going from i to i_h is $dQ_{i,i_h}(r)$. Once the X(t) process reaches either i_s or i_h, further transitions in it will be governed by cdf's, $G_{i_s,n}$ and $G_{i_h,n}$, respectively. Thus, the renewal equation for $G_{i,n}$, i = n+1, ...,N can be written as

$$G_{i,n}(t) = \sum_{k\epsilon E} \int_0^t G_{k,n}(t - x)\, dQ_{i,k}(x)$$

$$= Q_{i,i_h} * G_{i_h,n}(t) + Q_{i,i_s} * G_{i_s,n}(t)$$

$$= Q_{i,i_h} * Q_H * Q_{i_h,i} * G_{i,n}(t) + Q_{i,i_s} * Q_{i_s,i} * G_{i,n}(t)$$

$$+ Q_{i,i_s} * Q_{i_s,i-1} * G_{i-1,n}(t), \qquad \text{(A2.22)}$$

where E is the state space, $G_{n,n} = 1$, $Q_H = \sum_j Q^j_{i_h,i_h}$, and $Q^j_{i_h,i_h}$ is the j-fold convolution of Q_{i_h,i_h} with itself.

Taking the Laplace-Stieltjes (L-S) transform of Equation (A2.22) we get

$$\tilde{G}_{i,n}(s) = \tilde{Q}_{i,i_h}(s)\tilde{Q}_H(s)\tilde{Q}_{i_h,i}(s)\tilde{G}_{i,n}(s)$$

$$+ \tilde{Q}_{i,i_s}(s)\tilde{Q}_{i_s,i}(s)\tilde{G}_{i,n}(s)$$

$$+ \tilde{Q}_{i,i_s}(s)\tilde{Q}_{i_s,i-1}(s)\tilde{G}_{i-1,n}(s), \qquad \text{(A2.23)}$$

where

$$\tilde{Q}_H(s) = \sum_{j=0}^{\infty} \tilde{Q}^j_{i_h,i_h}(s) = \frac{s + \gamma}{s + p_h\gamma}$$

and, from Equations (A2.16) to (A2.21),

$$\tilde{Q}_{i,i_h}(s) = \frac{\beta}{s + \beta + \lambda_i},$$

$$\tilde{Q}_{i_h,i_h}(s) = \frac{q_h\gamma}{s + \gamma},$$

$$\tilde{Q}_{i_h,i}(s) = \frac{p_h\gamma}{s + \gamma},$$

$$\tilde{Q}_{i,i_s}(s) = \frac{\lambda_i}{s + \beta + \lambda_i},$$

$$\tilde{Q}_{i_s,i}(s) = \frac{q_s \mu_i}{s + \mu_i},$$

$$\tilde{Q}_{i_s,i-1}(s) = \frac{p_s \mu_i}{s + \mu_i}.$$

On substituting the expressions for the various L-S transforms in Equation (A2.23) and simplifying, we get

$$\tilde{G}_{i,n}(s) = a_i \tilde{G}_{i,n}(s) + b_i \tilde{G}_{i-1,n}(s), \tag{A2.24}$$

where

$$a_i = \frac{p_h\gamma\beta(s+\mu_i) + q_s\lambda_i\mu_i(s+p_h\gamma)}{(s+p_h\gamma)(s+\beta+\lambda_i)(s+\mu_i)}, \tag{A2.25}$$

$$b_i = \frac{p_s\lambda_i\mu_i}{(s+\mu_i)(s+\beta+\lambda_i)}. \tag{A2.26}$$

For $i = n+1$, $\tilde{G}_{i-1,n}(s) = \tilde{G}_{n,n}(s) = 1$, and

$$\tilde{G}_{i,n}(s) = \tilde{G}_{n+1,n}(s) = \frac{b_i}{1 - a_i} \tag{A2.27}$$

$$= \frac{p_s\lambda_i\mu_i(s+p_h\gamma)}{(s+x_{1,i})(s+x_{2,i})(s+x_{3,i})}, \tag{A2.28}$$

where $-x_{1,i}$, $-x_{2,i}$, and $-x_{3,i}$ are the roots of the polynomial

$$s^3 + s^2(\lambda_i+\mu_i+\beta+p_h\gamma) + s(p_s\lambda_i\mu_i+\beta\mu_i+\lambda_i p_h\gamma+\mu_i p_h\gamma) + p_s p_h\gamma\lambda_i\mu_i$$

$$\tilde{G}_{N,n}(s) = \prod_{i=n+1}^{N} \frac{p_s\lambda_i\mu_i\{(s+p_h\gamma)\}}{(s+x_{1,i})(s+x_{2,i})(s+x_{3,i})} \tag{A2.29}$$

Further, let

$$x_{1,n+1} = x_1$$

$$x_{2,n+1} = x_2$$

$$x_{3,n+1} = x_3$$

$$x_{1,n+2} = x_4 \qquad \text{(A2.30)}$$

$$x_{2,n+2} = x_5$$

$$x_{3,n+2} = x_6$$

$$x_{1,N} = x_{1(N-n)}$$

$$x_{2,N} = x_{2(N-n)}$$

$$\vdots$$

$$x_{3,N} = x_{3(N-n)}$$

and

$$K = 3(N-n)$$

$$\tilde{G}_{N,n}(s) = \frac{\{p_s(s+p_h\gamma)^{N-n} \prod_{i=n+1}^{N} \lambda_i\mu_i\}}{\prod_{j=1}^{K}(s+x_j)} . \qquad \text{(A2.31)}$$

By using the results from Lemma A2.3, the pdf and the cdf of $T_{N,n}$ are obtained from Equation (A2.31) as

$$g_{N,n}(t) = \sum_{j=1}^{K} \frac{U_{n+1}^{N}(-x_j+p_h\gamma)^{N-n}}{\prod_{\substack{i=1 \\ i\neq j}}^{K}(-x_j + x_i)} e^{-x_j t} \qquad \text{(A2.32)}$$

and

$$G_{N,n}(t) = \sum_{j=1}^{K} \frac{U_{n+1}^{N}(-x_j + p_h\gamma)^{N-n}}{\prod_{\substack{i=1 \\ i\neq j}}^{K}(-x_j + x_i)} \cdot \frac{1}{-x_j}(e^{-x_j t} - 1) \qquad \text{(A2.33)}$$

where

$$U_{n+1}^{N} = \prod_{i=n+1}^{N} (p_s \lambda_i \mu_i).$$

The distribution function of the first passage time to enter a state corresponding to a specified number of remaining software errors will be useful in the study and analysis of the other performance measures.

A2.2.2 Mean and Variance of $T_{N,n}$

Now,

$$E[T_{N,n}] = \int_0^{\infty} t g_{N,n}(t) dt \qquad \text{(A2.34)}$$

Substituting for $g_{N,n}(t)$ from Equation (A2.32), we get

$$E[T_{N,n}] = \sum_{j=1}^{K} \frac{U_{n+1}^{N} (-x_j + p_h\gamma)^{N-n}}{\prod_{\substack{i=1 \\ i\neq j}}^{K} (-x_j + x_i)} \cdot \int_0^{\infty} te^{-x_j t} dt$$

or

$$E[T_{N,n}] = \sum_{j=1}^{K} \frac{U_{n+1}^{N} (-x_j + p_h\gamma)^{N-n}}{\prod_{\substack{i=1 \\ i\neq j}}^{K} (-x_j + x_i)} \cdot \frac{1}{(x_j)^2} . \qquad \text{(A2.35)}$$

Similarly, to get the variance of $T_{N,n}$ we have

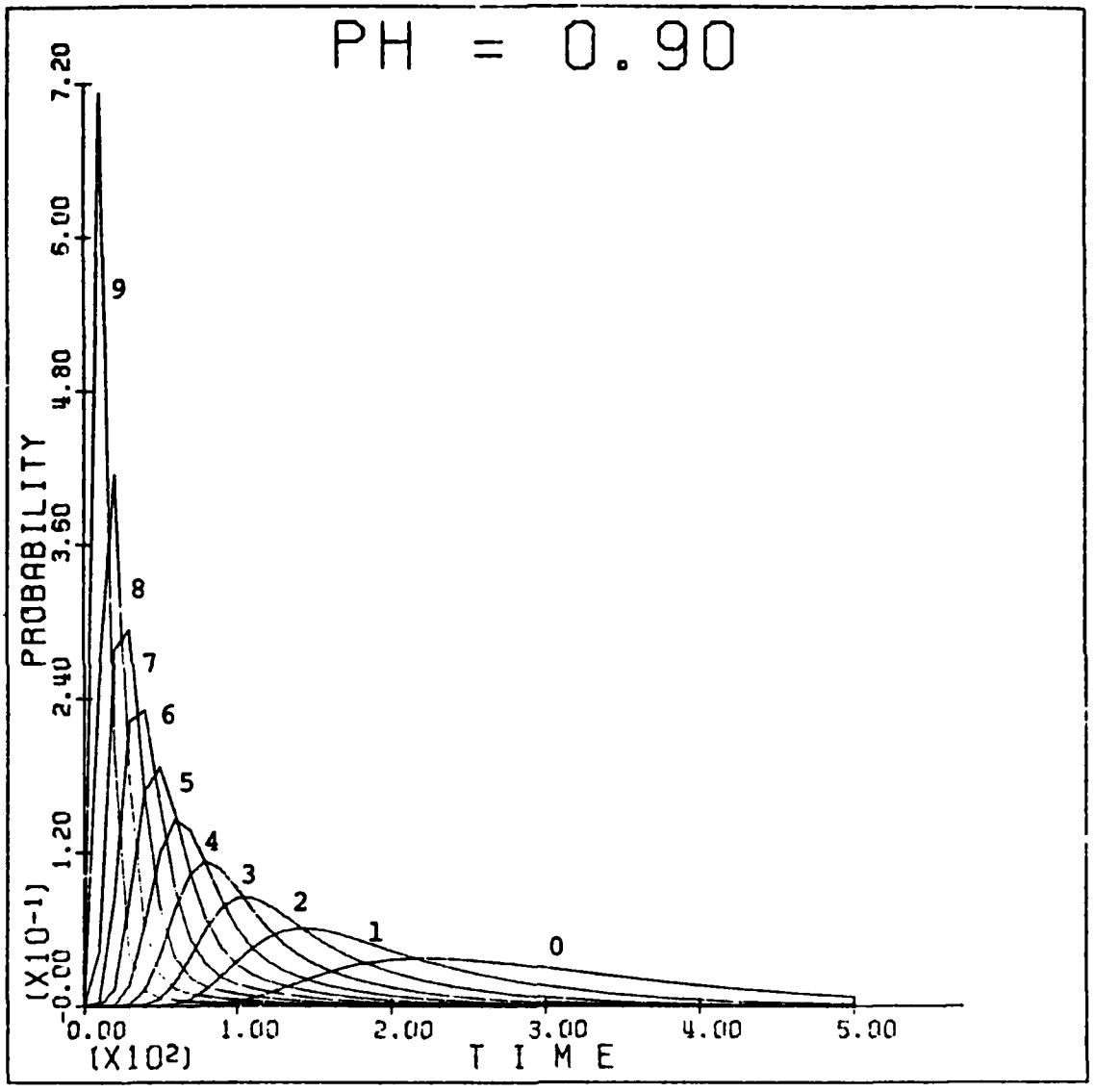
$$E[T_{N,n}^2] = \sum_{j=1}^{K} \frac{U_{n+1}^{N} (-x_j + p_h\gamma)^{N-n}}{\prod_{\substack{i=1 \\ i\neq j}}^{K} (-x_j + x_i)} \cdot \frac{2}{(x_j)^3} \qquad \text{(A2.36)}$$

and

$$Var[T_{N,n}] = E[T_{N,n}^2] - E^2[T_{N,n}]. \qquad \text{(A2.37)}$$

A2.2.2 Illustrative Example

Consider a system with N = 10 errors, p_s = 0.9, and p_h = 0.9. Assume that $\lambda_i = i\lambda$, $\mu_i = i\mu$, and the parametric values are λ = .02, μ = .05, β = .01, and γ = .025. We are interested in the distribution of $T_{N,n}$, n = 0,1,2,...,8,9. The pdf's and cdf's of $T_{N,n}$ for various values of n and for t from 0 to 500 units are computed from Equation (A2.32) and (A2.33), respectively, and are shown in Figures A2.4 and A2.5, respectively. Also, the means and variances of these distributions are obtained from Equations (A2.35) and (A2.37) respectively, and are summarized in Table A2.1. From Figures (A2.4) and (A2.5), we notice that the distributions are highly dependent on n. Also, as expected, the distribution of the time to an error-free software system has a large mean and a large variance (see Table A2.1). The mean and variance of $T_{10,n}$ for p_h = 1.0 are also given in Table A2.1. We note that both of these values are smaller

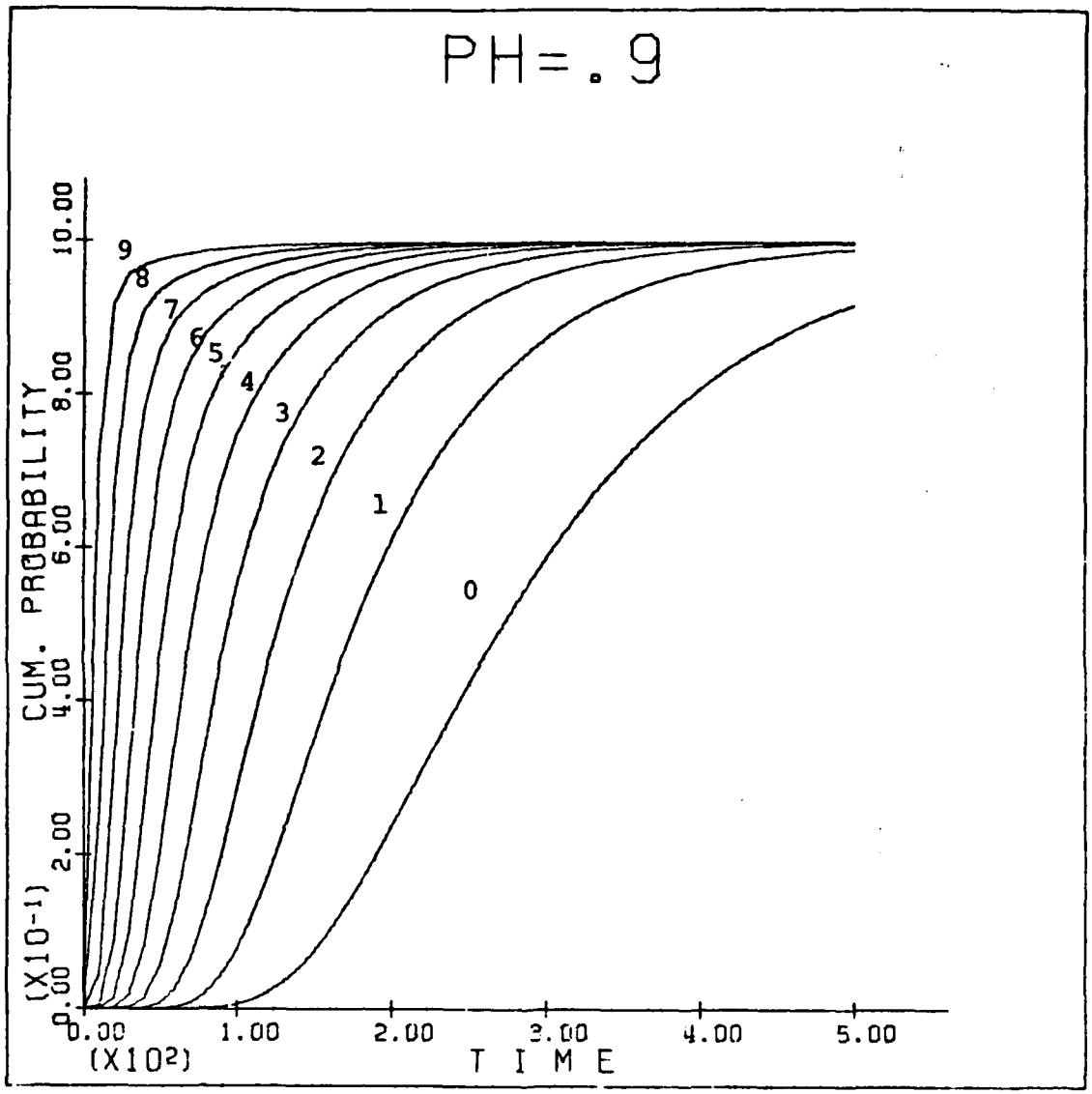


$\lambda_i = i\lambda$ $\lambda = .02$ $\beta = .01$ $p_s = .9$

$\mu_i = i\mu$ $\mu = .05$ $\gamma = .025$ $p_h = .9$

$N = 10$, $n = 9,8,\ldots,1,0$

Figure A2.4. Probability Distribution Function of First Passage Time to n.



$\lambda = .02$ $\beta = .01$ $p_s = .9$

$\mu = .05$ $\gamma = .025$ $p_h = .9$

$N = 10,$ $n = 9,8,7,\ldots,2,1,0$

Figure A2.5. Cumulative Distribution Function of First Passage Time to n.

TABLE A2.1

MEAN, VARIANCE, AND STANDARD DEVIATION
OF FIRST PASSAGE TIME DISTRIBUTION
N = 10.

TO STATE	PH = 0.9			PH = 1.0		
	MEAN	VARIANCE	S.D.	MEAN	VARIANCE	S.D.
9	10.2	292.4	17.1	10.0	246.7	15.7
8	21.6	626.2	25.0	21.1	529.2	23.0
7	34.4	1014.5	31.9	33.6	859.1	29.3
6	49.1	1476.8	38.4	47.9	1253.7	35.4
5	66.2	2045.1	45.1	64.7	1741.3	41.7
4	86.7	2775.7	52.7	84.6	2372.4	48.7
3	112.3	3780.0	61.5	109.6	3247.4	57.0
2	146.4	5321.6	72.9	142.9	4605.5	67.9
1	197.7	8241.5	90.8	192.9	7216.6	85.0
0	300.1	17726.3	133.1	292.9	15883.2	126.0

than those for $p_h = 0.9$ because of an improvement in the hardware system maintenance activity.

A2.3 State Occupancy Probabilities

In this Section we are interested in deriving expressions for the probability that the system is operational at time t with a specified number of remaining software errors. Let $P_{N,n}(t)$ be the probability that the system is operational at time t with n remaining software errors, given that it was in operation at time t = 0 with N software errors, i.e.

$$P_{N,n}(t) = P\{X(t)=n \mid X(0)=N\}, \quad n = 0,1,\ldots,N \tag{A2.38}$$

We call $P_{N,n}(t)$ the (operational) state occupancy probability. By conditioning on the first up-down cycle of the process and using an approach similar to that of Section A2.2 we get the following renewal equation for $P_{n,n}(t)$

$$P_{n,n}(t) = e^{-(\lambda_n+\beta)t} + Q_{n,n}*P_{n,n}(t) \tag{A2.39}$$

By conditioning on the first passage time, we get

$$P_{N,n}(t) = P_{n,n}*G_{N,n}(t). \tag{A2.40}$$

To obtain the L-S transform of $P_{N,n}(t)$, we take the L-S transforms of Equations (A2.39) and (A2.40) and solve the resulting equations. Let a_i's, b_i's, and $x_{i,j}$'s be as given in Equations (A2.25), (A2.26), and (A2.30), respectively.

Then, by having $\tilde{G}_{N,N}(s) = 1$, and letting

$$A(s) = s\beta(s+\mu_n) + \lambda_n(s+p_s\mu_n)(s+p_h\gamma), \text{ and}$$

$$B(s) = (s - x_{1,n})(s - x_{2,n})(s - x_{3,n}),$$

the L-S transforms of $P_{N,n}(t)$ and $P_{N,0}(t)$, respectively, are

$$\tilde{P}_{N,n}(s) = (1 - \frac{A(s)}{B(s)})\,\tilde{G}_{N,n}(s) \qquad \text{(A2.41)}$$

and

$$\tilde{P}_{N,0}(s) = (1 - \frac{\beta}{s + \beta + p_h\gamma})\tilde{G}_{N,0}(s)$$

The expressions for $P_{N,0}(t)$ and $P_{N,n}(t)$, $n = 1,\ldots,N$ are obtained from the results of Lemma A2.1 to A2.3 as

$$P_{N,n}(t) = G_{N,n}(t) - \sum_{j=1}^{K} \frac{U_{n+1}^{N}(-x_j + p_h\gamma)^{N-n}A(-x_j)}{\prod_{\substack{i=1 \\ i\neq j}}^{K}(-x_j + x_i)} \cdot \frac{(1 - e^{-x_j t})}{x_j}$$

and

$$P_{N,0}(t) = G_{N,0}(t) - \sum_{j=1}^{K_1} \frac{\beta U_1^N (-x_j + p_h\gamma)^N}{\prod_{\substack{i=1 \\ i\neq j}}^{K_1}(-x_j + x_i)} \cdot \frac{(1 - e^{-x_j t})}{x_j}$$

where $K = 3(N-n+1)$ and $K_1 = 3N+1$ are the number of roots in the denominator.

A2.4 System Reliability and Availability

A2.4.1 System Reliability

The reliability of a system at time x is given by

$$\bar{F}(x) = 1 - F(x)$$

where F is the life distribution of the system. The corresponding conditional reliability of a unit of age t is

$$\bar{F}(x|t) = \frac{\bar{F}(t+x)}{\bar{F}(t)}, \text{ if } \bar{F}(t) > 0$$

Consider our hardware-software system. At $t = 0$, the initial number of software errors in the system is equal to N. The reliability of the system at this stage is

$$\begin{aligned} P\{\text{up time} > x\} &= P\{\min(U, T_N) > x\} \\ &= P\{U > x\} \cdot P\{T > x\} \\ &= e^{-(\beta+\lambda_N)x} \end{aligned}$$

Next, consider some time $t > 0$ when the system has just been repaired and there are i remaining errors. The reliability of the system is

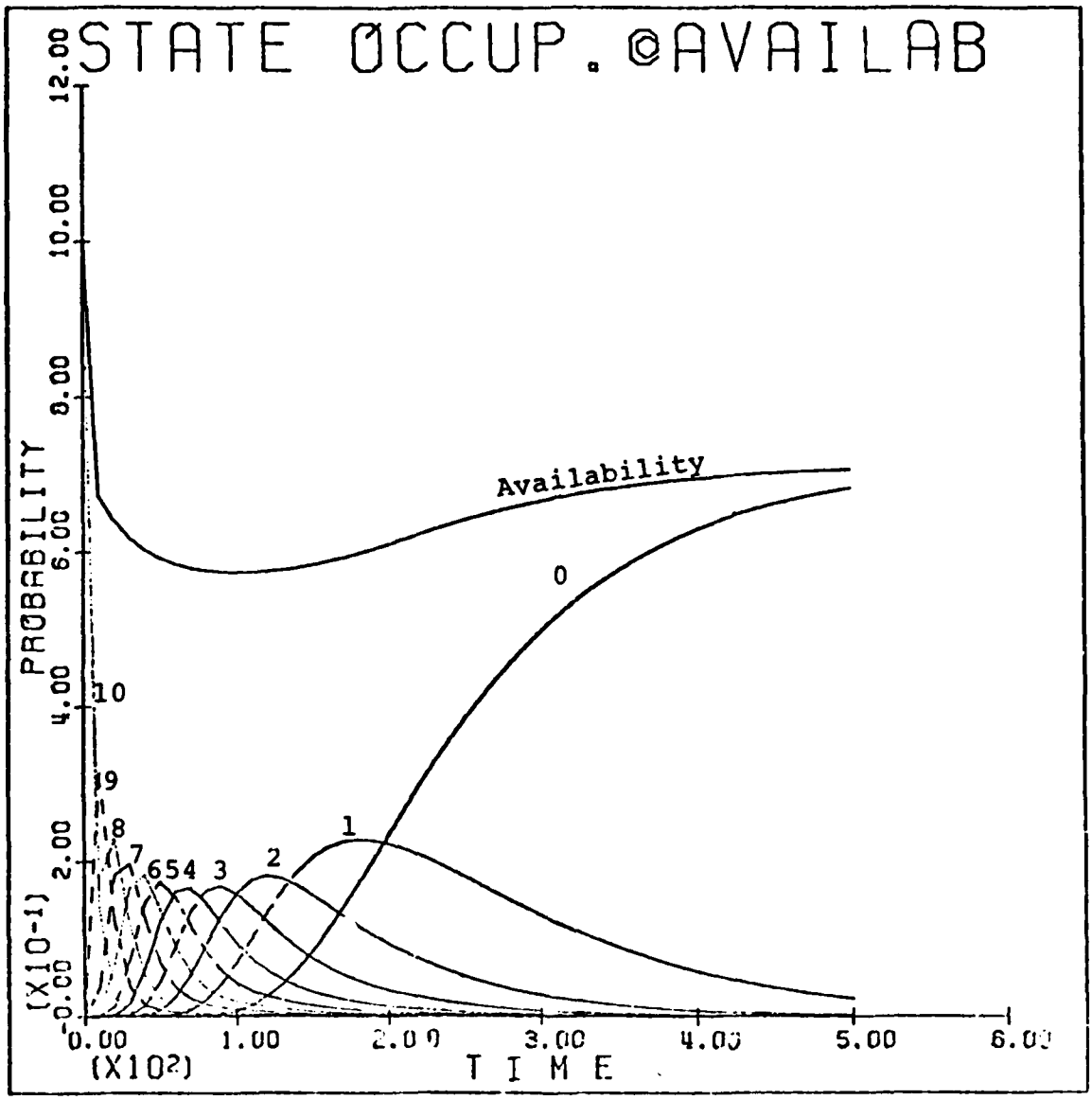
$$P\{\text{up time} > x | X(t) = i\} = P\{\min(U, T_i) > x\} = e^{-(\beta+\lambda_i)x} \quad \text{(A2.42)}$$

A2.4.2 System Availability

Another useful measure of system performance is its availability, which is defined as the probability that it is operational at some given time t. In our case, the system will be operational if the hardware system is in an up state and the software system is in an up state with n remaining errors, $n = 0, 1, \ldots, N$. In Section A2.3, we derive the expressions for $P_{N,n}(t)$, the probability that the system is operational at time t with n errors in the software system, given that it was operational at $t = 0$ with N software errors. Thus, the system availability can be defined as

$$A(t) = \sum_{n=0}^{N} P_{N,n}(t) \quad \text{(A2.43)}$$

To see the behavior of A(t) we consider an example with $N = 10$, $p_s = 0.9$, $p_h = 0.9$, $\lambda = .02$, $\mu = .05$, $\beta = .01$ and



$\lambda = .02$ $\quad \beta = .01$ $\quad p_s = .9$

$\mu = .05$ $\quad \gamma = .025$ $\quad p_h = .9$

$N = 10,$ $\quad n = 10,9,\ldots,1,0$

Figure A2.6. State Occupancy Probabilities and System Availability.

Table A2.2

Selected Values of $P_{N,n}(t)$ and A(t)

N = 10

n	Time 50	100	200	300	400	500
10	.003	.001	.000	.000	.000	.000
9	.009	.003	.000	.000	.000	.000
8	.030	.006	.001	.000	.000	.000
7	.081	.012	.002	.000	.000	.000
6	.149	.024	.004	.001	.000	.000
5	.166	.052	.008	.001	.000	.000
4	.106	.106	.018	.004	.001	.000
3	.036	.159	.042	.011	.003	.001
2	.006	.139	.105	.037	.011	.003
1	.000	.054	.215	.146	.074	.033
0	.000	.005	.191	.437	.578	.645
A(t)	.586	.561	.586	.637	.667	.682

$\gamma = .025$. For these values, distributions $P_{N,n}(t)$, $n = 0,1....,10$, t from 0 to 500 are obtained as described in Section A2.3. Selected values of $P_{N,n}(t)$ are given in Table A2.2 and the probability distributions are plotted in Figure A2.6 for $n=0,1,...,10$. The availability, as given by Equation (A2.43) is obtained as the sum of probabilities. Thus, for t = 100, we have

$$A(100) = \sum_{n=0}^{10} P_{10,n}(100) = 0.5612$$

Similarly

$$A(500) = \sum_{n=0}^{10} P_{10,n}(500) = 0.6819$$

Values of A(t) for various t are also plotted in Figure A2.6

A2.4.3 Average Availability

A sampling measure for the availability of an operational system is the ratio of total up time to total time elapsed. From a practical point of view, it is an important measurable sampling characteristic.

From the definition of availability, we find that the expected value of total up-time by time t can be expressed as

$$U(t) = \int_0^t A(x)dx.$$

The ratio of this value to the total time elapsed, t, will give us an average availability up to time t, $A_{av}(t)$, i.e.

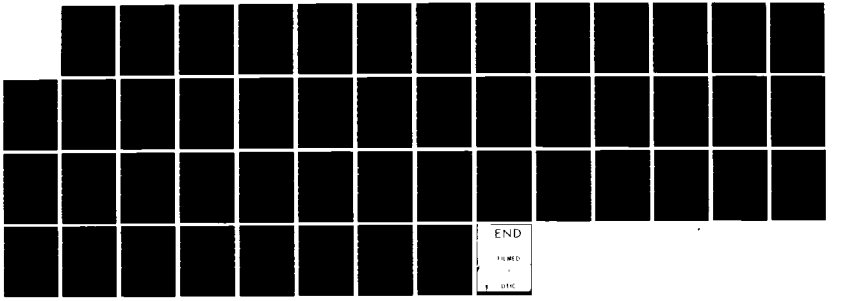
$$A_{av}(t) = \frac{U(t)}{t} = \frac{\int_0^t A(x)dx}{t}$$

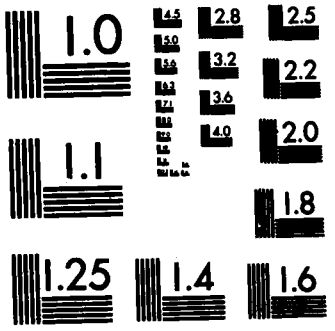
Similarly, the average unavailability can be expressed as

$$1 - A_{av}(t) = 1 - \frac{\int_0^t A(x)dx}{t} = \frac{\int_0^t [1 - A(x)]dx}{t}$$

AD-A123 421 SOFTWARE RELIABILITY MODELLING AND ESTIMATION TECHNIQUES(U) SYRACUSE UNIV N Y DEPT OF INDUSTRIAL ENGINEERING AND OPERATIONS RESEARCH A L GOEL OCT 82 4/4

UNCLASSIFIED RADC-TR-82-263 F30602-78-C-0351 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

A2.5 Expected Number of Software, Hardware and Total Failures by Time t

A2.5.1 Expected Number of Software Failures

Let $M_s(t)$ be the expected number of software failures detected by time t. In order to find the expression for $M_s(t)$, we consider a counting process $\{N_{si}(t), t \geq 0\}$, where $N_{si}(t)$ is the number of software failures detected during the time interval (0,t], when the initial number of errors in the software system is i. Let

$$M_{si}(t) = E[N_{si}(t) | X(0) = i] .$$

Then, by conditioning on the first passage time going from state N to i,

$$M_s(t) = \sum_{i=0}^{N} M_{si} * G_{N,i}(t) \qquad \text{(A2.45)}$$

where $M_{si}(t)$ can be obtained by conditioning on the first down cycle of the process

$$M_{si}(t) = Q_{i,i_s}(t) + Q_{i,i_s} * Q_{i_s,i} * M_{si}(t)$$

$$+ Q_{i,i_h} * Q_H * Q_{i_h,i} * M_{si}(t)$$

The Laplace Stieltjes transform of $M_{si}(t)$ is

$$\tilde{M}_{si}(s) = \frac{\lambda_i}{s + \beta + \lambda_i} + a_i \tilde{M}_{si}(s)$$

where a_i is defined in Equation (2.25). Now,

$$\tilde{M}_{si}(s) = \frac{\lambda_i}{s + \beta + \lambda_i} \frac{1}{1 - a_i}$$

or

$$\tilde{M}_{si}(s) = \frac{\lambda_i (s+\mu_i)(s+p_h\gamma)}{(s+x_{1,i})(s+x_{2,i})(s+x_{3,i})}$$

where $x_{1,i}$, $x_{2,i}$, $x_{3,i}$ are given in Equation (2.28).

In a simplified form,

$$\tilde{M}_{si}(s) = \frac{(s + \mu_i)}{p_s \mu_i} \tilde{G}_{i,i-1}(s) \qquad \text{(A2.46)}$$

From (A2.45) and (A2.46) the L-S transform for $M_s(t)$ is

$$\tilde{M}_s(s) = \sum_{i=1}^{N} \tilde{M}_{si} \tilde{G}_{N,i}(s)$$

$$= \sum_{i=1}^{N} \frac{(s + \mu_i)}{p_s \mu_i} \tilde{G}_{N,i-1}(s) \qquad \text{(A2.47)}$$

Finally, using Lemmas A2.1 to A2.3, we obtain the expression for $M_s(t)$ as

$$M_s(t) = \sum_{i=1}^{N} \sum_{j=1}^{K_i} \frac{U_i^N (-x_j + p_h\gamma)^{N-i+1} (-x_j + \mu_i)}{p_s \mu_i \prod_{\substack{\ell=1 \\ \ell \neq 1}}^{K_i} (-x_j + x_\ell)} \cdot \frac{(1 - e^{-x_j t})}{x_j}$$

where $K_i = 3(N-i+1)$.

A2.5.2 Expected Number of Hardware Failures

Let $M_h(t)$ be the expected number of hardware failures detected by time t. Consider a counting process, $\{N_{hi}(t), t \geq 0\}$, where $N_{hi}(t)$ is the number of hardware failures detected during the time interval (0,t], when the initial number of errors in the software system is i. Let

$$M_{hi}(t) = E[N_{hi}(t) | X(0) = i]$$

Then, by conditioning on the first passage time going from state N to i,

$$M_h(t) = \sum_{i=0}^{N} M_{hi} * G_{N,i}(t)$$

where $M_{hi}(t)$ can be obtained by conditioning on the first down cycle of the process

$$M_{hi}(t) = Q_{i,i_h}(t) + Q_{i,i_h} * Q_H * Q_{i_h,i} * M_{hi}(t)$$

$$+ Q_{i,i_s} * Q_{i_s,i} * M_{hi}(t)$$

Now, the L-S transform of $M_{hi}(t)$ for $i = 1,2,\ldots,N$ is

$$\tilde{M}_{hi}(s) = \frac{\beta}{s+\beta+\lambda_i} + a_i \tilde{M}_{hi}(s)$$

or

$$\tilde{M}_{hi}(s) = \frac{\beta(s+\mu_i)}{p_s \lambda_i \mu_i} \tilde{G}_{i,i-1}(s), \qquad \text{(A2.50)}$$

For $i = 0$, this L-S transform becomes

$$M_{h0}(s) = \frac{\beta}{s + \beta} \frac{1}{1 - a_0}$$

or

$$M_{h0}(s) = \frac{(s+p_h\gamma)}{s(s+\beta+p_h\gamma)} .$$

From (A2.49)the L-S transform for $M_h(t)$ is

$$\tilde{M}_h(s) = \sum_{i=0}^{N} \tilde{M}_{hi}(s) \tilde{G}_{N,i}(s) \qquad \text{(A2.51)}$$

The inverse L-S transform of $\tilde{M}_{hi}(s)\tilde{G}_{N,i}(s)$ is

$$G_i(t) = \sum_{j=1}^{K_i} \frac{\beta U_{i+1}^{N} (-x_j + p_h\gamma)^{N-i+1} (-x_j+\mu_i)}{\prod_{\substack{\ell=1 \\ \ell \neq j}}^{K_i} (-x_j + x_\ell)} \cdot \frac{(1 - e^{-x_j t})}{x_j}$$

where $K_i = 3(N-i+1)$, and the inverse L-S transform of $\tilde{M}_{h0}(s)\tilde{G}_{N,0}(s)$ is

$$G_0(t) = \frac{\beta U_1^N (p_h\gamma)^{N+1} t}{\prod_{\substack{j=1 \\ x_j \neq 0}}^{K-1} (x_j)} + \sum_{\substack{j=1 \\ x_j \neq 0}}^{K-1} \frac{\beta U_1^N (x_j + p_h\gamma)^{N+1}}{\prod_{\substack{\ell=1 \\ \ell \neq j}}^{K-1} (-x_j + x_\ell)} \cdot \frac{(1 - e^{-x_j t})}{x_j}$$

where $K = 3N + 2$.

Finally, the expression for $M_h(t)$ is

$$M_h(t) = G_0(t) + \sum_{i=1}^{N} G_i(t).$$

A2.5.3 Expected Number of Total Failures

Let $M(t)$ be the expected number of total failures detected by time t.

Consider $M_i(t)$ to be the expected number of total failures when there are i software errors in the system. For any $i = 0,1,2,\ldots,N$.

$$M_i(t) = M_{si}(t) + M_{hi}(t) \qquad \text{(A2.52)}$$

where $M_{s0}(t) = 0$,

and $\hat{M}_i(s) = \hat{M}_{si}(s) + \hat{M}_{hi}(s)$

Then

$$M(t) = \sum_{i=0}^{N} M_i * G_{N,i}(t), \qquad \text{(A2.53)}$$

$$\hat{M}(s) = \sum_{i=0}^{N} \hat{M}_i(s) \hat{G}_{N,i}(s)$$

or

$$\hat{M}(s) = \sum_{i=0}^{N} (\hat{M}_{si}(s) + \hat{M}_{hi}(s)) \hat{G}_{N,i}(s)$$

and

$$M(t) = M_s(t) + M_h(t) \qquad \text{(A2.54)}$$

TABLE A2.3

EXPECTED NUMBER OF FAILURES DETECTED PER TIME

TIME	SOFTWARE FAILURES	HARDWARE FAILURES	TOTAL FAILURES
0.00	0.00	0.00	0.00
20.00	2.48	0.14	2.62
40.00	4.19	0.26	4.45
60.00	5.47	0.38	5.85
80.00	6.47	0.49	6.96
100.00	7.27	0.61	7.87
120.00	7.92	0.72	8.64
160.00	8.91	0.94	9.85
200.00	9.59	1.17	10.77
240.00	10.06	1.41	11.48
280.00	10.39	1.66	12.05
320.00	10.62	1.92	12.53
360.00	10.77	2.18	12.95
400.00	10.88	2.44	13.32
440.00	10.95	2.71	13.66
480.00	11.00	2.98	13.99
520.00	11.04	3.25	14.29
560.00	11.06	3.53	14.59
600.00	11.08	3.80	14.88
640.00	11.09	4.08	15.17
680.00	11.10	4.35	15.45
720.00	11.10	4.63	15.73
760.00	11.10	4.91	16.01
800.00	11.11	5.18	16.29
840.00	11.11	5.46	16.57
880.00	11.11	5.74	16.85
920.00	11.11	6.01	17.12
960.00	11.11	6.29	17.40
1000.00	11.11	6.57	17.68

i.e. the expected number of total failures detected by time t is equal to the sum of the expected number of software and hardware failures.

A2.5.4 Illustrative Example

Consider a system with an initial number of software errors, N = 10; probabilities of perfect software and hardware maintenance $p_s = .9$ and $p_h = .9$, respectively; software failure rate $\lambda_i = i\lambda$, and $\lambda = .02$; software repair rate $\mu_i = i\mu$, and $\mu = .05$; and hardware failure and repair rates $\beta = .01$ and $\gamma = .025$, respectively.

For this system, the expected number of software, hardware, and total system failures are computed from Equations (A2.45), (A2.49), and (A2.54), respectively. Selected values of these quantities are given in Table A2.3 and plotted in Figure A2.7. The information in Table A2.3 shows us that the number of software failures detected is increasing rapidly at the early times and then slows down as the number of remaining software errors and software failure rate decrease.

On the other hand, the number of hardware failures detected is increasing with the slow-down of the number of software failures detected.

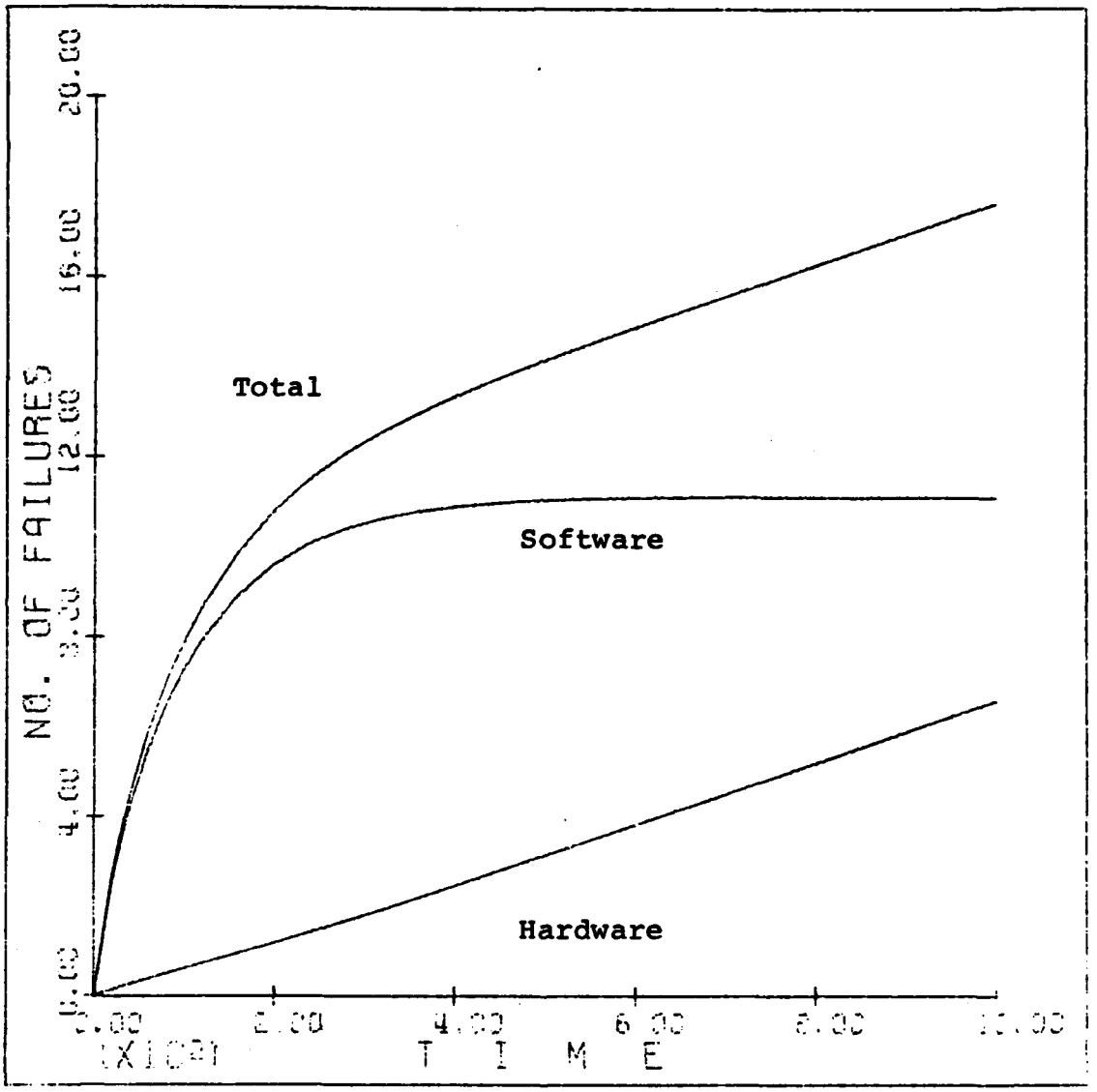


Figure A2.7. Expected Cumulative Number of Failures.

A3. OPERATIONAL COST MODELS

In Section 2 we proposed a model for the operational phase of the hardware-software system and developed expressions for several performance measures. In many applications, these individual measures are of less interest than an overall measure, such as the expected total cost. With this objective, in this section we develop cost models for the hardware, software, and hardware-software systems. The principal cost components considered are the cost of a failure, the cost of the maintenance activity performed to bring the system back to an operational state, and the cost of system downtime. The primary measures that affect the total cost are the number of failures and the system availability.

The relative importance of these measures in a given situation can be expressed via the numerical values for the cost factors.

Models for hardware and software systems are developed in Sections A3.1 and A3.2, respectively. The total hardware-software system is discussed in Section A3.3. Several numerical examples are used to illustrate the results.

A3.1 Operational Cost Model: Hardware System

In this Section we develop a cost model for the hardware system. The system is in an up state at time t = 0. After a random time U, whose distribution is exponential with parameter β (see Equation A2.5), a failure occurs and the system goes into a down state. A repair or maintenance activity is undertaken and after a random time V, whose distribution is exponential with parameter γ (see Equation A2.6), the system is brought back into an up state. The cause of the failure would have been removed with probability p_h $(0 \leq p_h \leq 1)$. The sequence of up and down states forms a renewal process. For purposes of this Section, it is assumed that the software system has no effect on the operation of the hardware system.

The following costs are incurred due to the failure and maintenance activities:

(i) A fixed cost c_{h_1} per failure

(ii) A variable cost c_{h_2} per repair per unit time

(iii) A variable cost due to the unavailability of the system, c_{h_3} per unit time.

Consider the time interval (0,t).

Let

$C_h(t)$ = expected total cost incurred by t,

$M_h(t)$ = expected number of hardware failures by t,

$A_h(t)$ = system availability at t.

Then, the expected total cost by time t is given by

$$C_h(t) = c_{h_1} M_h(t) + c_{h_2} \gamma t + c_{h_3} \int_0^t \{1-A_h(x)\}dx \quad (A3.1)$$

where

$\int_0^t \{1-A_h(x)\}dx$ is the expected total down time during (0,t). Now we develop expressions for $M_h(t)$ and $A_h(t)$ and obtain a closed form equation for $C_h(t)$. Consider one up and down cycle, i.e., one renewal. If the maintenance activity is perfect, the length of this cycle will be U + V. If, however, the maintenance activity is imperfect, the repair will go another V units of time so that the length of the cycle will be U + V + V. If the maintenance is imperfect for the second time, the length of the cycle will be U + V + V + V, and so on. Therefore, the probability density function, g, of the renewal time is given by

$$g = p_h f_U * f_V + p_h q_h f_U * f_V * f_V + p_h q_h^2 f_U * f_V * f_V * f_V + \ldots, \quad (A3.2)$$

where * stands for convolution,

f_U is the pdf of U ,

and f_V is the pdf of V.

The Laplace transform of g, g*, is

$$g^*(s) = p_h f_U^*(s) f_V^*(s) [1 + q_h f_V^*(s) + (q_h f_V^*(s))^2 + \ldots]$$

$$\text{or} \quad g^*(s) = p_h f_U^*(s) f_V^*(s) \left(\frac{1}{1 - q_h f_V^*(s)}\right)$$

$$\text{or} \quad g^*(s) = p_h \frac{\beta}{s + \beta} \frac{\gamma}{s + \gamma} \frac{s + \gamma}{s + p_h\gamma}$$

$$\text{or} \quad g^*(s) = \frac{\beta p_h \gamma}{(s + \beta)(s + p_h\gamma)} \tag{A3.3}$$

Now, the renewal equations for the expected number of hardware failures can be written as

$$M_h(t) = F_U(t) + \int_0^t M_h(t - x) g(x) dx \tag{A3.4}$$

where $F_U(t)$ is the cdf of U.

The Laplace transform of $M_h(t)$ is

$$M_h^*(s) = \frac{f_U^*(s)}{s} + M_h^*(s) g^*(s)$$

$$= \frac{\beta}{s(s + \beta)} + M_h^*(s) \frac{\beta}{s + \beta} \frac{p_h\gamma}{s + p_h\gamma}$$

or

$$M_h^*(s) = \frac{\beta(s + p_h\gamma)}{s^2(s + \beta + p_h\gamma)}$$

By taking the inverse Laplace transform we get

$$M_h(t) = \beta\left[\frac{1 - e^{-(\beta+p_h\gamma)t}}{\beta + p_h\gamma}\right] + \beta p_h\gamma\left[\frac{e^{-(\beta+p_h\gamma)t} - 1 + (\beta+p_h\gamma)t}{(\beta + p_h\gamma)^2}\right]$$

or

$$M_h(t) = \frac{\beta}{(\beta+p_h\gamma)^2}\left[p_h\gamma(\beta+p_h\gamma)t + \beta(1-e^{-(\beta+p_h\gamma)t})\right] \quad \text{(A3.5)}$$

The renewal equation for $A_h(t)$ can be written as

$$A_h(t) = 1 - F_U(t) + \int_0^t A_h(t - x)g(x)dx,$$

and its Laplace transform as

$$A_h^*(s) = \frac{1 - f_u^*(s)}{s[1 - g^*(s)]} = \frac{s + p_h\gamma}{s(s + \beta + p_h\gamma)}$$

Therefore, the availability of the system at time t is

$$A_h(t) = \mathcal{L}^{-1} A_h^*(s)$$

$$= e^{-(\beta+\gamma)t} + p_h\gamma\left[\frac{1 - e^{-(\beta+p_h\gamma)t}}{\beta + p_h\gamma}\right]$$

or

$$A_h(t) = \frac{p_h\gamma + \beta e^{-(\beta+p_h\gamma)t}}{\beta + p_h\gamma} \quad \text{(A3.6)}$$

Now, the expected total down time during (0,t) is

$\int_0^t \{1 - A_h(x)\}dx$ which, on substituting for $A_h(x)$ from Equation (A3.6), gives

$$\int_0^t (1 - A_h(x))dx = \beta\left[\frac{(\beta+p_h\gamma)t - 1 + e^{-(\beta+p_h\gamma)t}}{(\beta + p_h\gamma)^2}\right] \qquad (A3.7)$$

On substituting the expressions for $M_h(t)$ and $\int_0^t \{1-A_h(x)\}dx$ from Equations (A3.5) and (A3.7), respectively, in Equation (3.1), we get, after some simplification,

$$C_h(t) = \frac{c_{h_1}\beta}{(\beta+p_h\gamma)^2}\left[p_h\gamma(\beta+p_h\gamma)t + \beta(1-e^{-(\beta+p_h\gamma)t})\right] + c_{h_2}\gamma t$$

$$+ \frac{c_{h_3}\beta}{(\beta+p_h\gamma)^2}\left[(\beta+p_h\gamma)t - 1 + e^{-(\beta+p_h\gamma)t}\right] \qquad (A3.8)$$

The above equation gives the expected cost incurred by time t in terms of the hardware system parameters β, γ, and p_h, and the cost factors c_{h_1}, c_{h_2}, and c_{h_3}.

Illustrative Examples

We numerically study the behavior of $M_h(t)$, $A_{hav}(t)$ and $C_h(t)/t$ as a function of the cost factors c_{h_1}, c_{h_2}, c_{h_3} and of the failure and repair rates β and γ, respectively.

Consider a system with $\beta = .01$, $p_h = 0.9$, and $\gamma = .01$, .02, .05, .10, .20, .30, .50, .75, 1.0, 2.0, 3.0, and 4.0. The average availability ($A_{hav}(t)$) and the expected number of failures by time t are shown in Table A3.1 for t = 100, 250, 500, 1000, and 2000. We notice that for a fixed repair rate the average availability decreases with time, the rate of decrease being higher for low values of γ. The expected number of failures in a given time interval increases with γ. This is so because at low values of γ, the system is down for longer periods of time, causing a reduction in the up time of the system.

The expected total cost per unit time ($C_h(t)/t$) is now calculated from Equation (A3.8) for given cost factors. Such values for four sets of cost factors are given in Table A3.2. For a given t, the cost first decreases and then increases as a function of γ. In other words, $C_h(t)/t$ seems to be a convex function with respect to γ.

TABLE A3.1

AVERAGE AVAILABILITY AND EXPECTED NUMBER OF FAILURES
HARDWARE SYSTEM
FAILURE RATE: 0.010

AVERAGE AVAILABILITY

REPAIR RATE	T I M E 100.0	250.0	500.0	1000.0	2000.0
0.01	0.709261	0.583529	0.529082	0.501385	0.487535
0.02	0.762652	0.693831	0.668367	0.655612	0.649235
0.05	0.851105	0.831405	0.824793	0.821488	0.819835
0.10	0.910000	0.904000	0.902000	0.901000	0.900500
0.20	0.950139	0.948476	0.947922	0.947645	0.947507
0.30	0.965561	0.964796	0.964541	0.964413	0.964349
0.50	0.978733	0.978450	0.978355	0.978308	0.978284
0.75	0.985615	0.985487	0.985444	0.985423	0.985412
1.00	0.989132	0.989059	0.989035	0.989023	0.989017
1.50	0.992701	0.992669	0.992658	0.992652	0.992650
2.00	0.994506	0.994487	0.994481	0.994478	0.994477
3.00	0.996324	0.996315	0.996313	0.996311	0.996311
4.00	0.997238	0.997233	0.997231	0.997231	0.997230

EXPECTED NUMBER OF HARDWARE FAILURES

0.01	0.7093	1.4588	2.6454	5.0139	9.7507
0.02	0.7627	1.7346	3.3418	6.5561	12.9847
0.05	0.8511	2.0785	4.1240	8.2149	16.3967
0.10	0.9100	2.2600	4.5100	9.0100	18.0100
0.20	0.9501	2.3712	4.7396	9.4765	18.9501
0.30	0.9656	2.4120	4.8227	9.6441	19.2870
0.50	0.9787	2.4461	4.8918	9.7831	19.5657
0.75	0.9856	2.4637	4.9272	9.8542	19.7082
1.00	0.9891	2.4726	4.9452	9.8902	19.7803
1.50	0.9927	2.4817	4.9633	9.9265	19.8530
2.00	0.9945	2.4862	4.9724	9.9448	19.8895
3.00	0.9963	2.4908	4.9816	9.9631	19.9262
4.00	0.9972	2.4931	4.9862	9.9723	19.9446

TABLE A3.2

EXPECTED TOTAL COST PER UNIT TIME
HARDWARE SYSTEM
FAILURE RATE; 0.010

CH1=10, CH2=10, AND CH3=10

REPAIR RATE	TIME 100.0	250.0	500.0	1000.0	2000.0
0.01	3.0783	4.3231	4.8621	5.1363	5.2734
0.02	2.6497	3.3311	3.5832	3.7094	3.7726
0.05	2.0741	2.2691	2.3345	2.3673	2.3836
0.10	1.9910	2.0504	2.0702	2.0801	2.0850
0.20	2.5936	2.6101	2.6156	2.6183	2.6197
0.30	3.4409	3.4485	3.4510	3.4523	3.4529
0.50	5.3105	5.3133	5.3143	5.3147	5.3150
0.75	7.7424	7.7437	7.7441	7.7443	7.7444
1.00	10.2076	10.2083	10.2086	10.2087	10.2087
1.50	15.1723	15.1726	15.1727	15.1727	15.1728
2.00	20.1544	20.1546	20.1546	20.1547	20.1547
3.00	30.1364	30.1365	30.1365	30.1365	30.1365
4.00	40.1273	40.1274	40.1274	40.1274	40.1274

CH1=100, CH2=10, AND CH3=10

0.01	3.7167	4.8482	5.3383	5.5875	5.7122
0.02	3.3361	3.9555	4.1847	4.2995	4.3569
0.05	2.8401	3.0174	3.0769	3.1066	3.1215
0.10	2.8100	2.8640	2.8820	2.8910	2.8955
0.20	3.4488	3.4637	3.4687	3.4712	3.4724
0.30	4.3099	4.3168	4.3191	4.3203	4.3209
0.50	6.1914	6.1940	6.1948	6.1952	6.1954
0.75	8.6295	8.6306	8.6310	8.6312	8.6313
1.00	11.0978	11.0985	11.0987	11.0988	11.0988
1.50	16.0657	16.0660	16.0661	16.0661	16.0662
2.00	21.0494	21.0496	21.0497	21.0497	21.0497
3.00	31.0331	31.0332	31.0332	31.0332	31.0332
4.00	41.0249	41.0249	41.0249	41.0249	41.0249

TABLE A3.2
(CONTINUED)

EXPECTED TOTAL COST PER UNIT TIME
HARDWARE SYSTEM
FAILURE RATE: 0.010

CH1=10, CH2=10, AND CH3=100

REPAIR RATE	TIME 100.0	250.0	500.0	1000.0	2000.0
0.01	29.2448	41.8055	47.2447	50.0116	51.3953
0.02	24.0111	30.8863	33.4301	34.7043	35.3415
0.05	15.4747	17.4426	18.1031	18.4334	18.5985
0.10	10.0910	10.6904	10.8902	10.9901	11.0401
0.20	7.0812	7.2472	7.3025	7.3302	7.3441
0.30	6.5404	6.6169	6.6424	6.6551	6.6615
0.50	7.2245	7.2529	7.2623	7.2670	7.2694
0.75	9.0371	9.0499	9.0541	9.0563	9.0573
1.00	11.1857	11.1930	11.1954	11.1966	11.1972
1.50	15.8292	15.8324	15.8335	15.8340	15.8343
2.00	20.6489	20.6507	20.6513	20.6516	20.6518
3.00	30.4673	30.4681	30.4684	30.4685	30.4686
4.00	40.3760	40.3764	40.3766	40.3767	40.3767

CH1=10, CH2=100, AND CH3=10

0.01	3.9783	5.2231	5.7621	6.0363	6.1734
0.02	4.4497	5.1311	5.3832	5.5094	5.5726
0.05	6.5741	6.7691	6.8345	6.8673	6.8836
0.10	10.9910	11.0504	11.0702	11.0801	11.0851
0.20	20.5936	20.6101	20.6156	20.6183	20.6197
0.30	30.4409	30.4485	30.4510	30.4523	30.4529
0.50	50.3105	50.3133	50.3143	50.3147	50.3150
0.75	75.2424	75.2437	75.2441	75.2443	75.2444
1.00	100.2076	100.2083	100.2086	100.2087	100.2087
1.50	150.1723	150.1726	150.1727	150.1727	150.1728
2.00	200.1544	200.1546	200.1546	200.1547	200.1547
3.00	300.1364	300.1365	300.1365	300.1365	300.1365
4.00	400.1273	400.1274	400.1274	400.1274	400.1274

Plots of $A_{h_{av}}(t)$ for β = .01, .05, and 0.10 versus γ are shown in Figure A3.1. As expected, the average availability improves with γ as well as with an improvement in the failure rate, i.e., as β goes from 0.10 to 0.01.

Costs per unit times for various t are shown in Figure A3.2 for β = .01, c_{h_1} = 10, c_{h_2} = 10, c_{h_3} = 100 as a function of γ and clearly show the convexity of the cost function. A similar pattern is seen in Figure A3.3 which gives the plots of $C_h(t)/t$ for the four sets of cost factors at time t = 500 and β = .01.

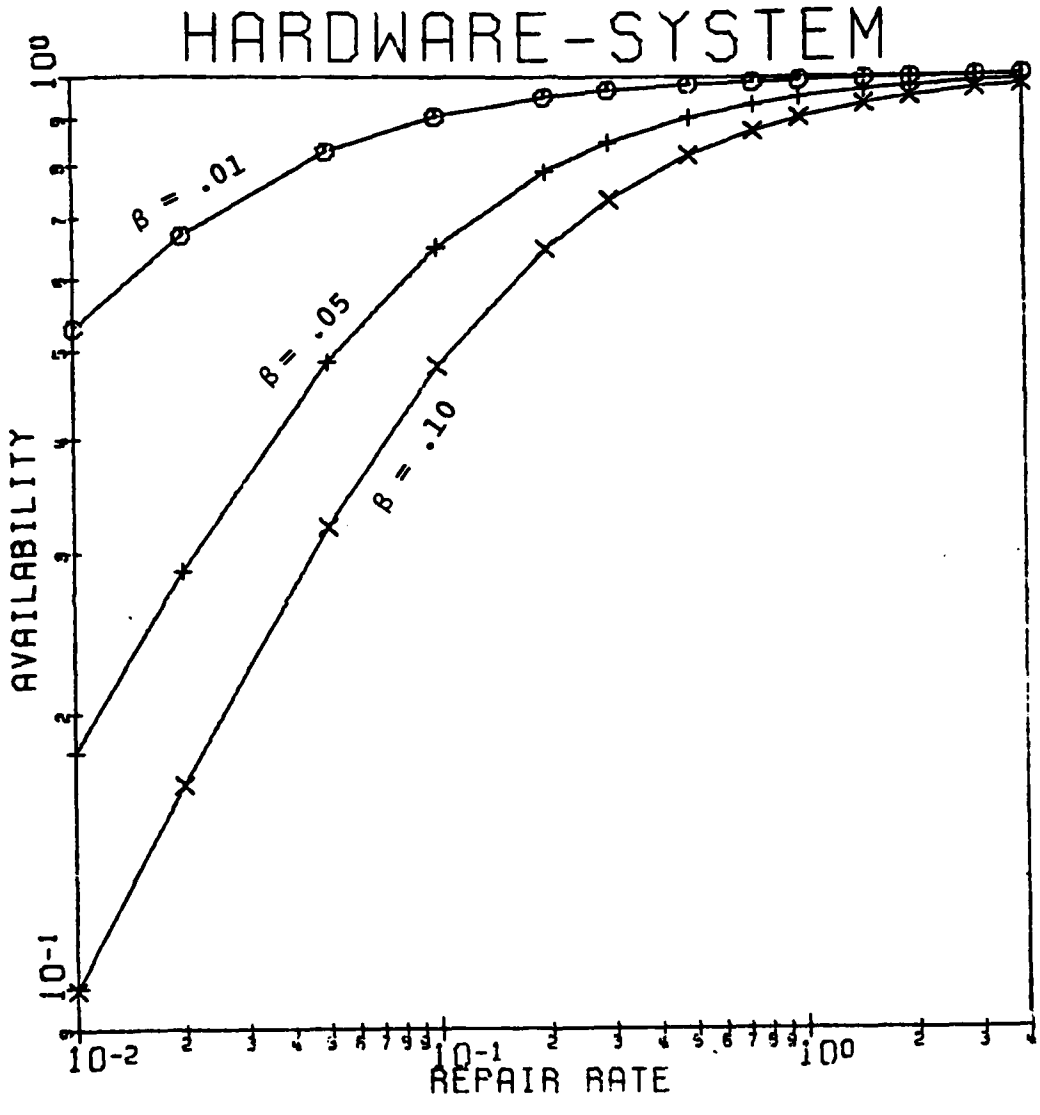


Figure A3.1. Average Availability vs. Repair Rate for Different Failure Rates. (t = 500)

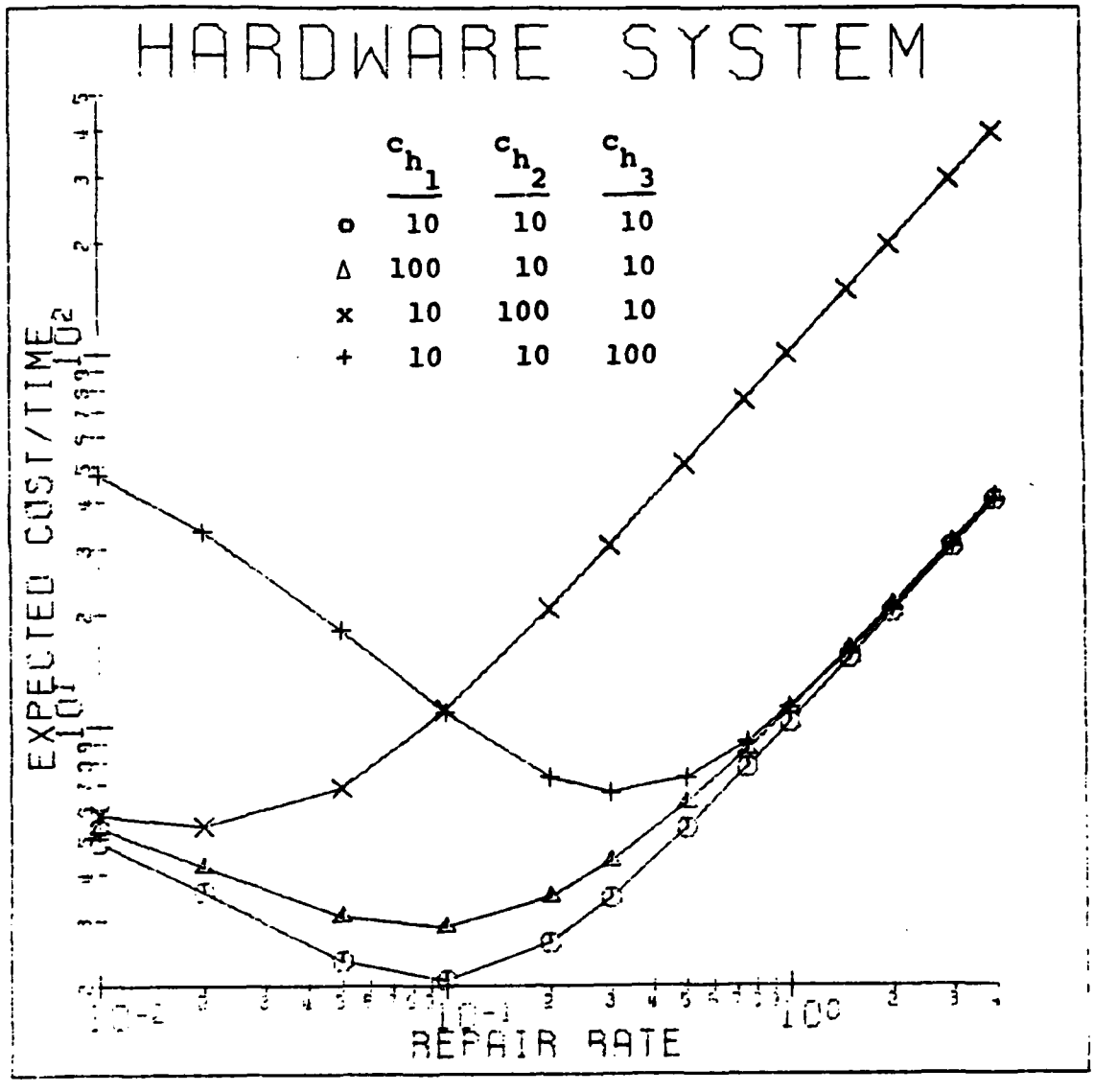


Figure A3.2. Expected Total Cost/Unit Time vs. Repair Rate for Different Cost Factors ($\beta = .01$, $t = 500$).

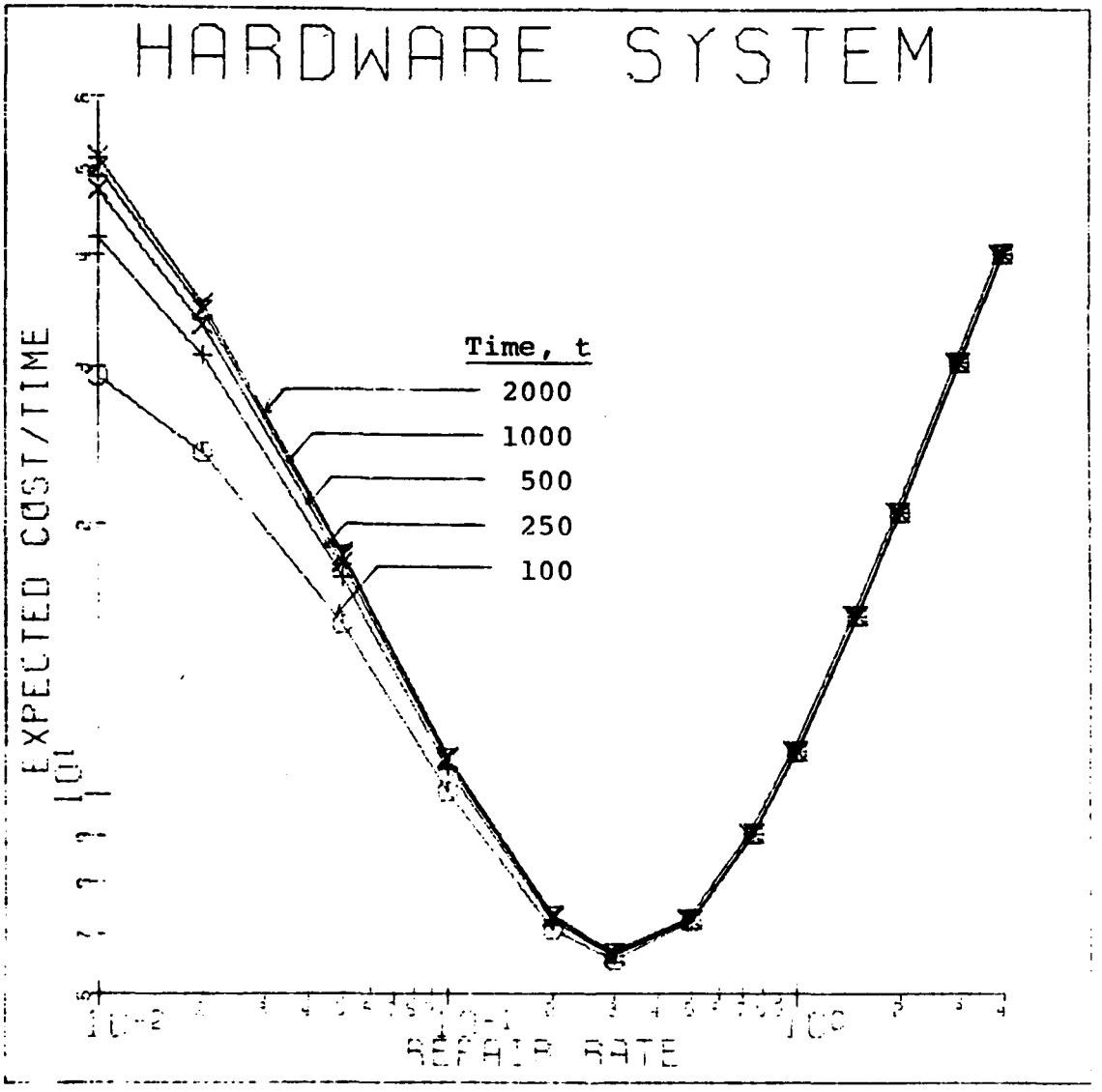


Figure A3.3. Expected Total Cost/Unit Time vs. Repair Rate for Different Times (β = .01, c_{h_1} = 10, c_{h_2} = 10, c_{h_3} = 100).

A3.2 Operational Cost Model: Software System

Consider a system consisting of software only. At time zero it is operational with N errors in the system. A failure occurs at a random time, T_N, whose distribution is given by Equation A2.2 with parameter λ_N. A repair is undertaken and with probability p_s the error causing the failure is removed in a time W_N whose distribution is exponential with parameter μ_N (see Equation A2.4.) The next cycle starts with (N-1) errors in the system and the failure distribution is now exponential with a parameter $(N-1)\lambda$. If the error is not removed, which happens with probability $q_s = 1 - p_s$, the distribution of time to next failure is again exponential with parameter λ_N. A similar behavior is observed throughout the entire life cycle of the software system with $i (0 \leq i \leq N)$ remaining errors. Note that the model is similar to the Imperfect Maintenance Model (IMM) of Okumoto and Goel (1978).

A diagrammatic representation of the behavior of the software system is shown in Figure A3.4.

As discussed in Section A3.1, for a hardware system, the cost elements associated with the failure-repair cycles of the software system are

c_{s_1} = cost of a software failure,

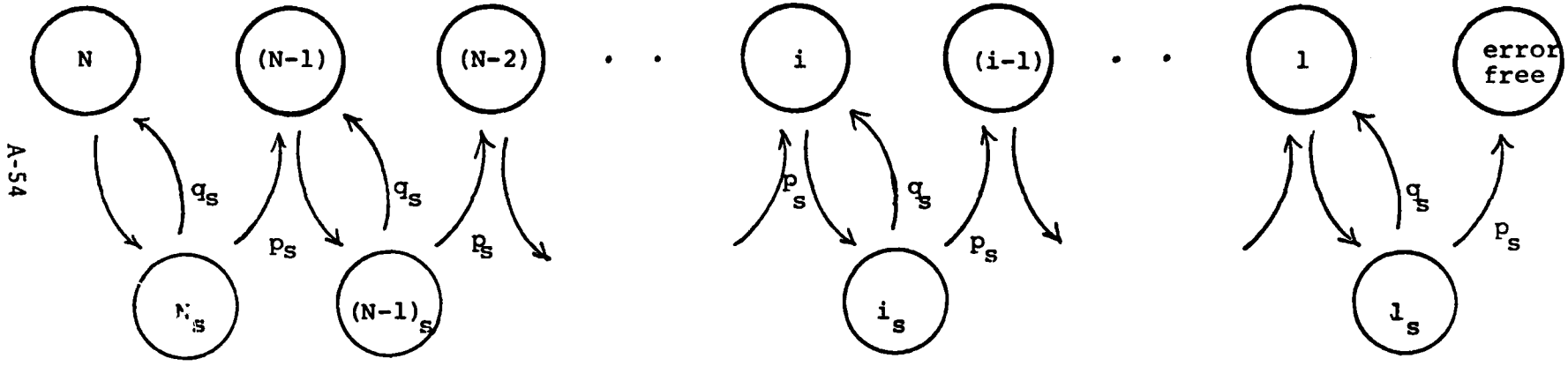


Figure A3.4. Software System: Diagrammatic Representation of Transitions between States of X(t).

c_{s_2} = cost incurred per repair per unit time,

and c_{s_3} = cost of system down time per unit time.

Then, the expected total operational cost by time t is given by

$$C_s(t) = c_{s_1} M_s(t) + c_{s_2} \mu t + c_{s_3} \int_0^t \{1 - A_s(x)\}\, dx \qquad \text{(A3.9)}$$

where

$M_s(t)$ = expected number of software failures by time t,

$A_s(t)$ = availability of the software system at time t.

To get the expression for $M_s(t)$ and $A_s(t)$, we first give the Laplace-Stieltjes transforms of the appropriate quantities as follows.

Let $G_{N,i}(t)$ be the distribution function of the first passage time from state N to state i. By considering the renewal equation associated with this, the Laplace-Stieltjes transform of $G_{N,i}(t)$ is obtained as

$$\tilde{G}_{N,i}(s) = \prod_{j=i+1}^{N} \frac{p_s \lambda_j \mu_j}{s^2 + s(\lambda_j + \mu_j) + p\lambda_j \mu_j} \qquad \text{(A3.10)}$$

Similarly, the L-S transforms of $M_s(t)$ and $A_s(t)$ are given by

$$\tilde{M}_s(s) = \sum_{i=1}^{N} \frac{\lambda_i (s + \mu_i)}{s^2 + s(\lambda_i + \mu_i) + p_s \lambda_i \mu_i} \tilde{G}_{N,i}(s) \qquad \text{(A3.11)}$$

and

$$\tilde{A}_s(s) = \sum_{i=0}^{N} \left(1 - \frac{\lambda_i(s + p_s\mu_i)}{s^2 + s(\lambda_i+\mu_i) + p_s\lambda_i\mu_i}\right) \tilde{G}_{N,i}(s) \qquad \text{(A3.12)}$$

where $\tilde{G}_{N,N}(s) = 1$.

Unlike the hardware system discussed in the previous Section, the results for $M_s(t)$ and $A_s(t)$ cannot be obtained in a closed form, but can be derived from Equations (A3.11) and (A3.12) by using Lemmas A2.1, A2.2, and A2.3 as follows.

First we obtain the inverse Laplace-Stieltjes transform for Equation (A3.10). We write

$$\tilde{G}_{N,i}(s) = \prod_{j=i+1}^{N} \frac{p_s\lambda_j\mu_j}{(s + x_{1,j})(s + x_{2,j})}$$

Let $x_{1,i+1} = x_1$, $x_{2,i+1} = x_2$, $x_{1,i+2} = x_3$, $x_{2,i+2} = x_4$, ..., $x_{2,N} = x_{K_i}$, where $K_i = (N-i) \times 2$,

and let

$$U_i^N = \prod_{j=i+1}^{N} p_s\lambda_j\mu_j,$$

as given in Equation (A2.33)

By Lemmas A2.1-A2.3

$$G_{N,i}(t) = \sum_{j=1}^{K_i} \frac{\left(\prod_{k=i+1}^{N} p_s\lambda_k\mu_k\right)}{\prod_{\substack{\ell=1 \\ j\neq\ell}}^{K_i} (-x_j + x_\ell)} \cdot \frac{(e^{-x_j t} - 1)}{-x_j}$$

$$= U_i^N \sum_{j=1}^{K_i} \frac{(e^{-x_j t} - 1)}{-x_j \prod_{\substack{\ell=1 \\ j\neq\ell}}^{K_i} (-x_j + x_\ell)}$$

Similarly, we get

$$\tilde{M}_s(s) = \sum_{i=1}^{N} \frac{\lambda_i(s+\mu_i)}{(s+x_{1,i})(s+x_{2,i})} \prod_{j=i+1}^{N} \frac{(p_s\lambda_j\mu_j)}{(s+x_{1,j})(s+x_{2,j})}$$

$$= \sum_{i=1}^{N} \left\{ \frac{\lambda_i(s+\mu_i)\ U_i^N}{\prod_{k=i}^{N}(s+x_{1,k})(s+x_{2,k})} \right\}$$

and

$$M_s(t) = \sum_{i=1}^{N} \left\{ \sum_{j=1}^{K_i} \left[\frac{\lambda_i \left(\prod_{k=i+1}^{N} p_s\lambda_k\mu_k \right)}{\prod_{\substack{\ell=1 \\ j\neq\ell}}^{K_i} (-x_j + x_\ell)} (e^{-x_j t} - 1) \right.\right.$$

$$\left.\left. + \frac{\lambda_i\mu_i \left(\prod_{k=i+1}^{N} p_s\lambda_k\mu_k \right)}{\prod_{\substack{\ell=1 \\ j\neq\ell}}^{K_i} (-x_j + x_\ell)} \cdot \frac{(e^{-x_j t} - 1)}{-x_j} \right] \right\}$$

or

$$M_s(t) = \sum_{i=1}^{N} \lambda_i\ U_i^N \left\{ \sum_{j=1}^{K_i} \frac{(x_j + \mu_i)(e^{-x_j t} - 1)}{-x_j \prod_{\substack{\ell=1 \\ j\neq\ell}}^{K_i} (-x_j + x_\ell)} \right\} \qquad \text{(A3.13)}$$

For the availability, taking the inverse L-S transform of Equation (A3.12), we have

$$A_s(t) = \sum_{i=0}^{N} \{G_{N,i}(t) - G_{N,i-1}(t) - \lambda_i U_i^N \sum_{j=1}^{K_i} \frac{(e^{-x_j t} - 1)}{\prod_{\substack{\ell=1 \\ j \neq \ell}}^{K_i} (-x_j + x_\ell)}\} \quad \text{(A3.14)}$$

For given c_{s_1}, c_{s_2}, and c_{s_3}, the expected total cost can be obtained from Equation (A3.9) by substituting for $M_s(t)$ and $A_s(t)$ from Equations (A3.13) and (A3.14), respectively.

Illustrative Examples

Now we numerically study the behavior of $M_s(t)$, $A_{sav}(t)$ and $C_s(t)/t$ as a function of the software repair rate μ, failure rate λ, and the cost factors c_{s_1}, c_{s_2}, and c_{s_3}.

Let us consider a system with N = 10, μ = 0.05, and p_s = 0.9. The values of $A_{sav}(t)$ and $M_s(t)$ computed from the formulae derived earlier in this section are given in Table 3.3 for various values of μ and t. From the table we note that the average availability improves with t as well as with μ. The improvement with t is due to the fact that, as more software errors are removed, the system fails less often. The improvement with repair rate is due to shorter down time.

The expected number of failures increases with t and

with μ. As the system is used for longer time, more errors surface resulting in more failures. Also, as μ improves, the system is up for longer periods of time resulting in more software failures. Note that the asymptotic value of $M_s(t)$ is simply the ratio $N/p_s = 10/.9 = 11.1111$. Plots of $A_{sav}(t)$ for $\lambda = .01$, .05, and .10 versus μ are shown in Figure 3.5. As one would expect, availability improves as μ goes up and also as λ goes from 0.10 to 0.05 to 0.01.

The expected total cost per unit time $C_s(t)/t$, is given in Table A3.4 for $\lambda = 0.05$, $N = 10$, $p_s = 0.9$, μ varying from 0.01 to 4.00, t from 100 to 2000, and the cost factors varying as follows:

c_{s_1}	c_{s_2}	c_{s_3}
10	10	10
100	10	10
10	10	100
10	100	10

Two additional sets of plots of the cost values versus μ, taken from the above tables, are shown in Figures A3.6 and A3.7.

TABLE A3.3

AVERAGE AVAILABILITY AND EXPECTED NUMBER OF FAILURES
SOFTWARE SYSTEM
FAILURE RATE: 0.050

AVERAGE AVAILABILITY

REPAIR RATE	TIME 100.0	250.0	500.0	1000.0	2000.0
0.01	0.196539	0.215310	0.392714	0.674846	0.837280
0.02	0.322907	0.424598	0.675202	0.837280	0.918640
0.05	0.548578	0.740689	0.869824	0.934912	0.967456
0.10	0.722236	0.869873	0.934912	0.967456	0.983728
0.20	0.849874	0.934920	0.967456	0.983728	0.991864
0.30	0.898108	0.956612	0.978304	0.989152	0.994576
0.50	0.938155	0.973966	0.986982	0.993491	0.996746
0.75	0.958565	0.982644	0.991322	0.995661	0.997830
1.00	0.968853	0.986983	0.993491	0.996746	0.998373
1.50	0.979190	0.991322	0.995661	0.997830	0.998915
2.00	0.984376	0.993491	0.996746	0.998373	0.999186
3.00	0.989574	0.995661	0.997830	0.998915	0.999458
4.00	0.992176	0.996746	0.998373	0.999186	0.999593

EXPECTED NUMBER OF SOFTWARE FAILURES

0.01	6.6245	10.1158	11.0700	11.1109	11.1111
0.02	8.6796	11.0015	11.1106	11.1111	11.1111
0.05	10.3617	11.1095	11.1111	11.1111	11.1111
0.10	10.8006	11.1109	11.1111	11.1111	11.1111
0.20	10.9340	11.1110	11.1111	11.1111	11.1111
0.30	10.9650	11.1110	11.1111	11.1111	11.1111
0.50	10.9858	11.1110	11.1111	11.1111	11.1111
0.75	10.9949	11.1110	11.1111	11.1111	11.1111
1.00	10.9992	11.1110	11.1111	11.1111	11.1111
1.50	11.0034	11.1110	11.1111	11.1111	11.1111
2.00	11.0054	11.1111	11.1111	11.1111	11.1111
3.00	11.0073	11.1111	11.1111	11.1111	11.1111
4.00	11.0083	11.1111	11.1111	11.1111	11.1111

TABLE A3.4

EXPECTED TOTAL COST PER UNIT TIME
SOFTWARE SYSTEM
FAILURE RATE: 0.050

CS1=10, CS2=10, AND CS3=10

REPAIR RATE	TIME 100.0	250.0	500.0	1000.0	2000.0
0.01	8.7971	8.3515	6.3943	3.4626	1.7828
0.02	7.8389	6.3941	3.6702	1.9383	1.0692
0.05	6.0504	3.5375	2.0240	1.2620	0.8810
0.10	4.8577	2.7457	1.8731	1.4366	1.2183
0.20	4.5947	3.0952	2.5477	2.2738	2.1369
0.30	5.1154	3.8783	3.4392	3.2196	3.1098
0.50	6.7170	5.7048	5.3524	5.1762	5.0881
0.75	9.0138	8.1180	7.8090	7.6545	7.5773
1.00	11.4114	10.5746	10.2873	10.1437	10.0718
1.50	16.3084	15.5312	15.2656	15.1328	15.0664
2.00	21.2568	20.5095	20.2548	20.1274	20.0637
3.00	31.2050	30.4878	30.2439	30.1220	30.0610
4.00	41.1791	40.4770	40.2385	40.1192	40.0596

CS1=100, CS2=10, AND CS3=10

0.01	14.7591	11.9932	8.3869	4.4626	2.2828
0.02	15.6505	10.3546	5.6701	2.9383	1.5692
0.05	15.3759	7.5369	4.0240	2.2620	1.3810
0.10	14.5783	6.7456	3.8731	2.4366	1.7183
0.20	14.4352	7.0952	4.5477	3.2738	2.6369
0.30	14.9839	7.8783	5.4392	4.2196	3.6098
0.50	16.6042	9.7048	7.3524	6.1762	5.5881
0.75	18.9093	12.1180	9.8090	8.6545	8.0773
1.00	21.3107	14.5746	12.2873	11.1437	10.5718
1.50	26.2115	19.5312	17.2656	16.1328	15.5664
2.00	31.1616	24.5095	22.2548	21.1274	20.5637
3.00	41.1116	34.4878	32.2439	31.1220	30.5610
4.00	51.0865	44.4770	42.2385	41.1192	40.5596

TABLE A3.4
(CONTINUED)

EXPECTED TOTAL COST PER UNIT TIME
SOFTWARE SYSTEM
FAILURE RATE: 0.050

CS1=10, CS2=10, AND CS3=100

REPAIR RATE	TIME 100.0	250.0	500.0	1000.0	2000.0
0.01	81.1085	78.9736	61.0500	32.7265	16.4276
0.02	68.7773	58.1803	32.9020	16.5832	8.3916
0.05	46.6783	26.8755	13.7398	7.1199	3.8100
0.10	29.8565	14.4571	7.7310	4.3655	2.6828
0.20	18.1060	8.9525	5.4766	3.7383	2.8692
0.30	14.2857	7.7833	5.3918	4.1959	3.5980
0.50	12.2831	8.0478	6.5240	5.7620	5.3810
0.75	12.7429	9.6800	8.5901	8.0450	7.7725
1.00	14.2146	11.7461	10.8731	10.4366	10.2183
1.50	18.1813	16.3122	15.6561	15.3281	15.1640
2.00	22.6629	21.0953	20.5477	20.2738	20.1369
3.00	32.1434	30.8783	30.4392	30.2196	30.1098
4.00	41.8832	40.7699	40.3849	40.1925	40.0962

CS1=10, CS2=100, AND CS3=10

0.01	9.6971	9.2515	7.2943	4.3626	2.6828
0.02	9.6389	8.1941	5.4702	3.7383	2.8692
0.05	10.5504	8.0375	6.5240	5.7620	5.3810
0.10	13.8577	11.7457	10.8731	10.4366	10.2183
0.20	22.5947	21.0952	20.5477	20.2738	20.1369
0.30	32.1154	30.8783	30.4392	30.2196	30.1098
0.50	51.7170	50.7048	50.3524	50.1762	50.0881
0.75	76.5138	75.6180	75.3090	75.1545	75.0773
1.00	101.4114	100.5746	100.2873	100.1437	100.0718
1.50	151.3084	150.5312	150.2656	150.1328	150.0664
2.00	201.2568	200.5095	200.2548	200.1274	200.0637
3.00	301.2050	300.4878	300.2439	300.1220	300.0610
4.00	401.1791	400.4770	400.2385	400.1192	400.0596

Figure A3.6 shows how the average cost changes with μ for the four sets of cost factors. The minimum for these plots occurs at different values of μ due to changes in the cost factors. In Figure A3.7, the average costs are shown for various time horizons. We note that all the curves follow a similar pattern, i.e., first decreasing with μ and then increasing.

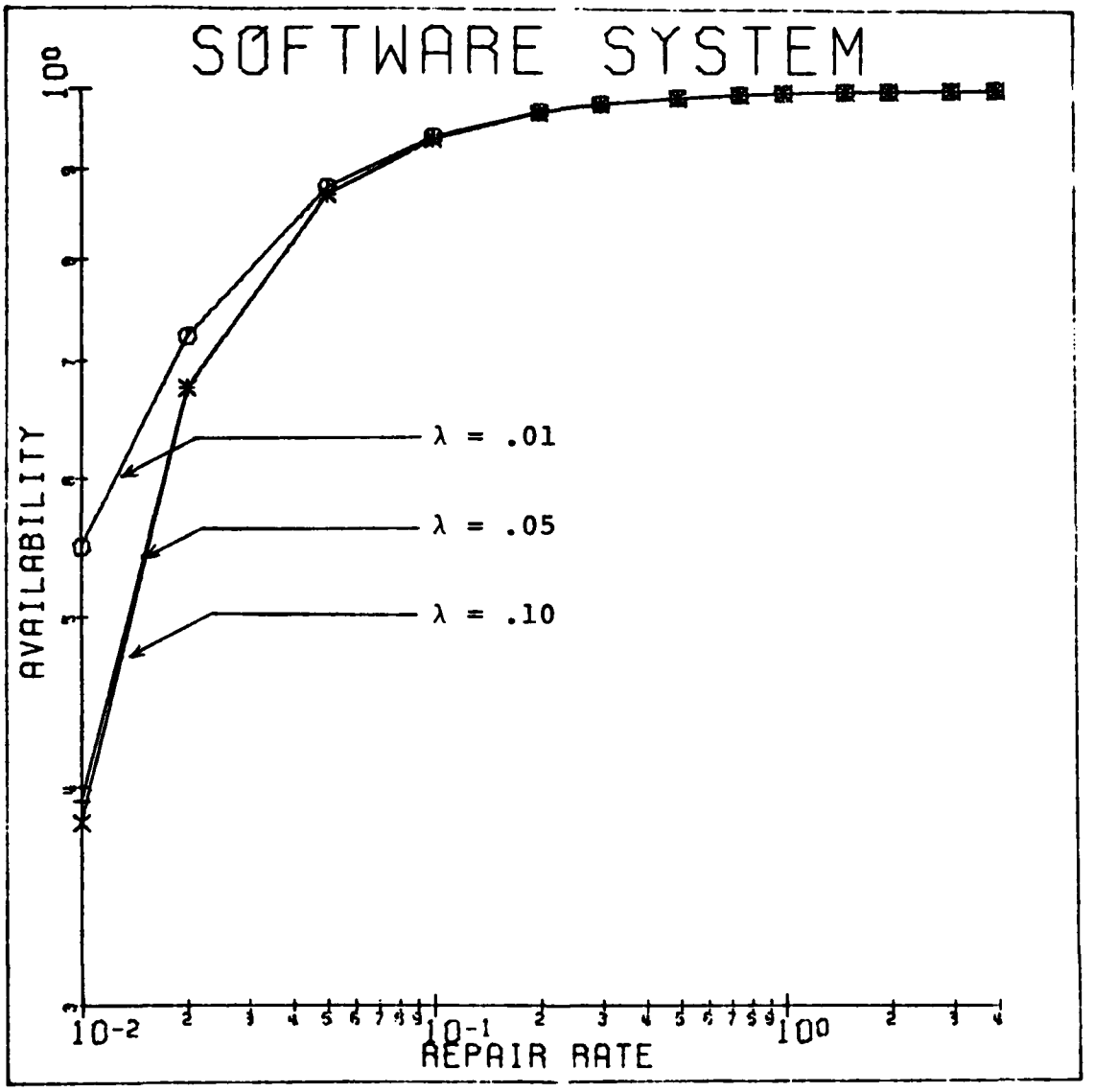


Figure A3.5. Average Availability vs. Repair Rate for Different Failure Rates (t = 500).

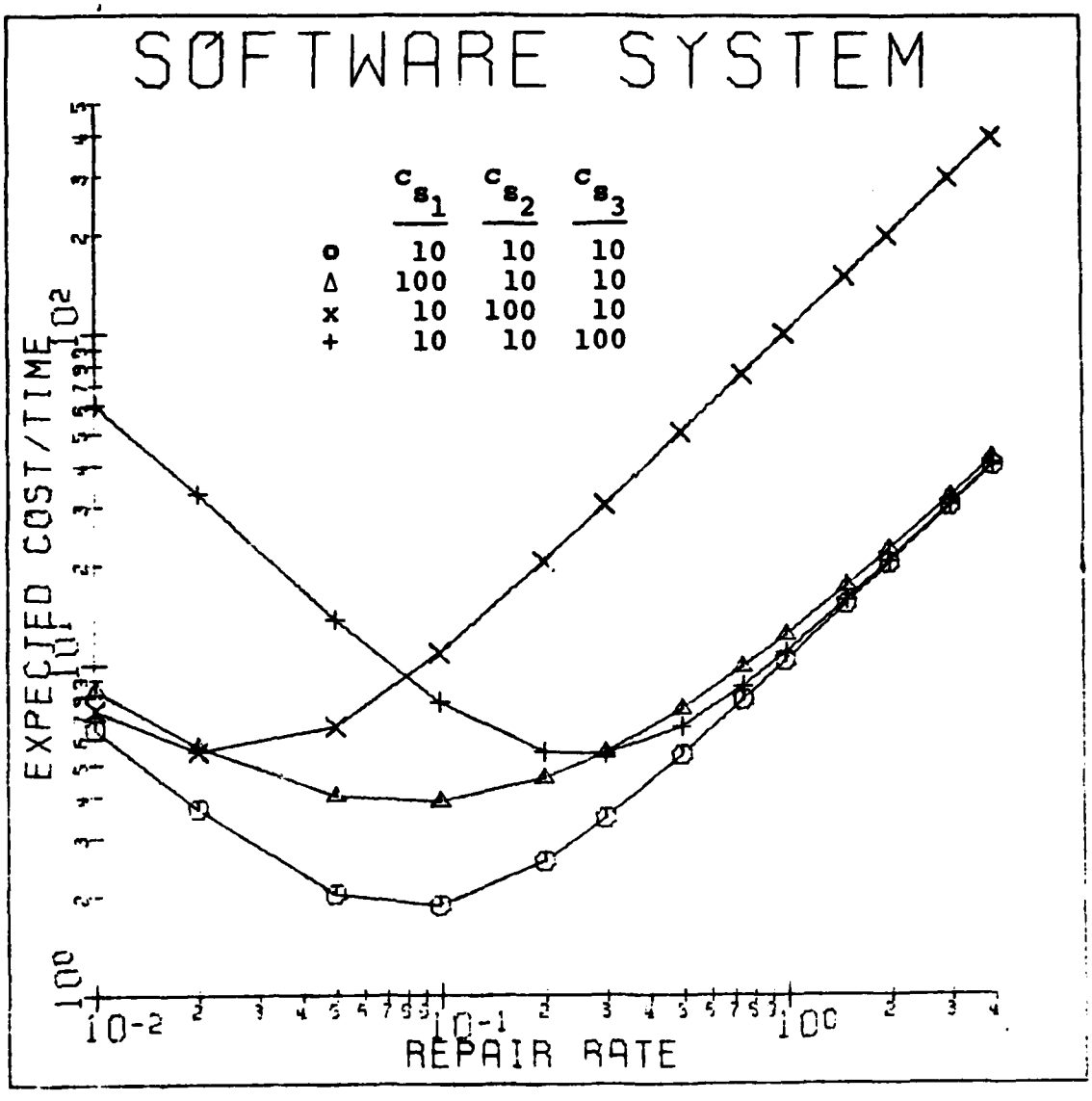


Figure A3.6. Expected Total Cost/Unit Time vs. Repair Rate for Different Cost Factors ($\lambda = .05$, $t = 500$).

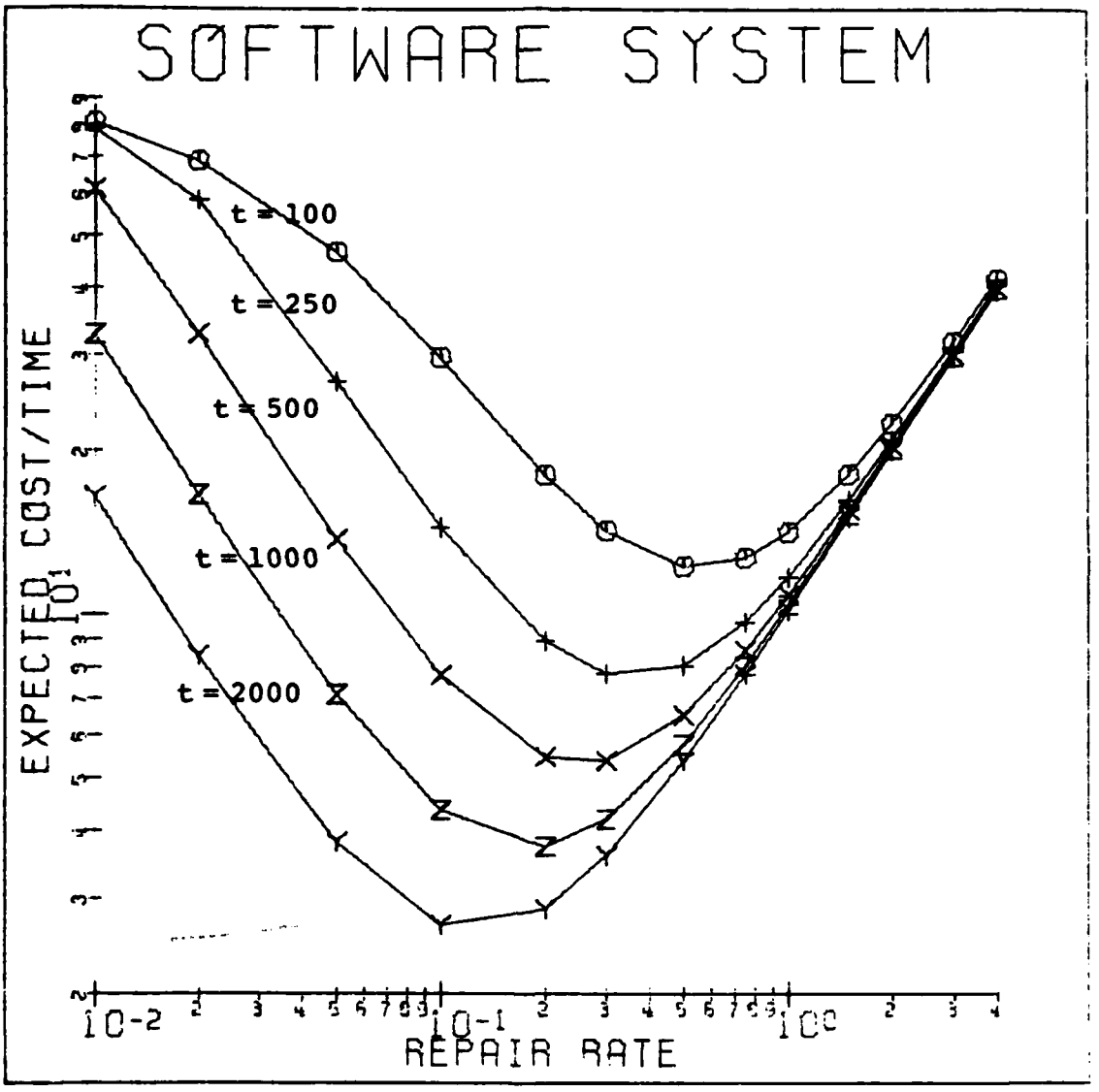


Figure A3.7. Expected Total Cost/Unit Time vs. Repair Rate for Different Times ($\lambda = .05$, $c_{s_1} = 10$, $c_{s_2} = 10$, $c_{s_3} = 10$)).

A3.3 Operational Cost Model: Hardware-Software System

We consider a hardware-software system whose behavior is the same as the system discussed in Section A2.1. Having discussed the cost models for hardware only and software only systems in the previous Sections, the operational cost of the hardware-software system is basically the sum of both the operational costs. However, the performance measures required in the hardware-software system are not obviously equal to their respective sums.

The cost elements associated with the operation of this system are c_{h_1}, c_{h_2}, and c_{h_3} as defined in Section A3.1, and c_{s_1}, c_{s_2}, and c_{s_3}, as defined in Section A3.2.

The performance measures required for the cost model are: $M_h(t)$ and $M_s(t)$, the expected number of hardware and software failures by time t, respectively, and the expected total down time during $(0,t)$.

Let $C(t)$ be the expected total operational cost associated with the hardware-software system and let $c_{h_3} = c_{s_3} = c_3$. Then

$$C(t) = c_{h_1} M_h(t) + c_{s_1} M_s(t) + c_{h_2} \gamma t + c_{s_2} \mu t + c_3 \int_0^t (1 - A(x))dx, \qquad \text{(A3.15)}$$

where the expressions for $M_h(t)$, $M_s(t)$, and $A(\cdot)$ are given in Equations (A2.49), (A2.54), and (A2.43), respectively.

Illustrative Examples

Now we numerically study the behavior of $M_h(t)$, $M_s(t)$, $C(t)/t$ and $A_{av}(t)$ as a function of γ and μ. The values of $A(x)$, $A_{av}(t)$, $M_s(t)$ and $M_h(t)$ are computed from Equations (A2.43), (A2.44), (A2.45), and (A2.49), respectively. The values of $C(t)$ are given by Equation (A3.15).

Let us consider a system with $N = 10$, $p_s = .9$, $p_h = .9$, $\beta = .01$, and $\lambda = .05$. For $t = 100$, $\gamma = .02$ to 1.0 and $\mu = 0.01$ to 0.50, the values of $A_{av}(t)$, $M_s(t)$ and $M_h(t)$ are given in Table A3.5. We note that the average availability improves with both the hardware and the software repair rates. Also, the expected number of failures increases with increase in γ and μ. This is because of the increased amount of time that the system is up leading to a longer time available for the failures to occur. Note that for these data sets all software errors have been removed by approximately $t = 500$. As pointed out earlier, after this happens, the system behaves as a hardware only system. To see how $C(t)/t$ behaves as a function of γ and μ, let us suppose that $c_{s_1} = 10$, $c_{h_1} = 10$, $c_{s_2} = 10$, $c_{h_2} = 10$, $c_{s_3} = c_{h_3} = 10$, and $t = 100$ to 2000 as shown in Table A3.9. As can be easily seen, the cost varies with both γ and μ. As an example, for $t = 500$, $\mu = 0.10$, $C(t)/t$ goes from 5.97 to 12.23 as γ goes from 0.02 to 1.00. The

minimum seems to occur around $\gamma = 0.10$. Similarly, for $\gamma = 0.10$, $t = 500$, $C_s(t)$ goes from 9.07 to 7.63 as μ goes from 0.01 to 0.50 with the minimum occurring at around $\mu = 0.10$. A similar behavior is seen for other t values.

TABLE A3.5

AVERAGE AVAILABILITY AND EXPECTED NUMBER OF FAILURES,
HARDWARE-SOFTWARE SYSTEM

(t = 100)

AVERAGE AVAILABILITY

SOFTWARE REPAIR RATE	HARDWARE REPAIR RATE 0.020	0.050	0.100	0.200	0.500	1.000
0.010	0.1814651	0.1866103	0.1897074	0.1916580	0.1929772	0.1934441
0.050	0.4625249	0.4938450	0.5143947	0.5283556	0.5383869	0.5420751
0.100	0.5831032	0.6329669	0.6664243	0.6896547	0.7066430	0.7129479
0.250	0.6842970	0.7542586	0.8014733	0.8342371	0.8580026	0.8667412
0.500	0.7216969	0.8003935	0.8532790	0.8897114	0.9159166	0.9254982

EXPECTED NUMBER OF SOFTWARE FAILURES

0.010	6.2889	6.4384	6.5168	6.5609	6.5885	6.5979
0.050	9.5311	9.9464	10.1424	10.2383	10.2913	10.3080
0.100	9.9296	10.4010	10.6042	10.6928	10.7368	10.7498
0.250	10.1058	10.6000	10.7944	10.8697	10.9036	10.9131
0.500	10.1547	10.6534	10.8423	10.9121	10.9423	10.9506

EXPECTED NUMBER OF HARDWARE FAILURES

0.010	0.1840	0.1892	0.1923	0.1943	0.1956	0.1961
0.050	0.4650	0.4965	0.5170	0.5310	0.5410	0.5447
0.100	0.5856	0.6356	0.6690	0.6923	0.7093	0.7156
0.250	0.6868	0.7569	0.8041	0.8369	0.8606	0.8694
0.500	0.7242	0.8030	0.8559	0.8923	0.9185	0.9281

TABLE A3.6

AVERAGE AVAILABILITY AND EXPECTED NUMBER OF FAILURES, HARDWARE-SOFTWARE SYSTEM

(t = 500)

AVERAGE AVAILABILITY

SOFTWARE REPAIR RATE	HARDWARE REPAIR RATE 0.020	0.050	0.100	0.200	0.500	1.000
0.010	0.2838317	0.3315687	0.3573419	0.3733835	0.3842965	0.3881781
0.050	0.5842655	0.7177649	0.7843187	0.8240747	0.8504862	0.8597665
0.100	0.6260339	0.7710168	0.8428978	0.8857371	0.9141593	0.9241394
0.250	0.6511171	0.8029690	0.8780454	0.9227346	0.9523632	0.9627631
0.500	0.6594803	0.8136197	0.8897613	0.9350671	0.9650979	0.9756377

EXPECTED NUMBER OF SOFTWARE FAILURES

0.010	10.9891	11.0493	11.0589	11.0623	11.0641	11.0646
0.050	11.1041	11.1111	11.1111	11.1111	11.1111	11.1111
0.100	11.1060	11.1111	11.1111	11.1111	11.1111	11.1111
0.250	11.1069	11.1111	11.1111	11.1111	11.1111	11.1111
0.500	11.1071	11.1111	11.1111	11.1111	11.1111	11.1111

EXPECTED NUMBER OF HARDWARE FAILURES

0.010	1.4218	1.6605	1.7893	1.8695	1.9241	1.9435
0.050	2.9239	3.5914	3.9242	4.1230	4.2550	4.3014
0.100	3.1328	3.8577	4.2171	4.4313	4.5734	4.6233
0.250	3.2582	4.0175	4.3928	4.6163	4.7644	4.8164
0.500	3.3000	4.0707	4.4514	4.6779	4.8281	4.8808

TABLE A3.7

AVERAGE AVAILABILITY AND EXPECTED NUMBER OF FAILURES,
HARDWARE-SOFTWARE SYSTEM

(t = 1000)

AVERAGE AVAILABILITY

SOFTWARE REPAIR RATE	HARDWARE REPAIR RATE 0.020	0.050	0.100	0.200	0.500	1.000
0.010	0.4464477	0.5552353	0.6081232	0.6393556	0.6599666	0.6671837
0.050	0.6135084	0.7679722	0.8421592	0.8857214	0.9143735	0.9243887
0.100	0.6344296	0.7945992	0.8714489	0.9165528	0.9462101	0.9565752
0.250	0.6469823	0.8105754	0.8890227	0.9350515	0.9653120	0.9758870
0.500	0.6511665	0.8159008	0.8948806	0.9412177	0.9716794	0.9823243

EXPECTED NUMBER OF SOFTWARE FAILURES

0.010	11.1100	11.1106	11.1107	11.1107	11.1108	11.1108
0.050	11.1111	11.1111	11.1111	11.1111	11.1111	11.1111
0.100	11.1111	11.1111	11.1111	11.1111	11.1111	11.1111
0.250	11.1111	11.1111	11.1111	11.1111	11.1111	11.1111
0.500	11.1111	11.1111	11.1111	11.1111	11.1111	11.1111

EXPECTED NUMBER OF HARDWARE FAILURES

0.010	4.4671	5.5550	6.0838	6.3962	6.6023	6.6745
0.050	6.1377	7.6823	8.4242	8.8598	9.1463	9.2465
0.100	6.3469	7.9486	8.7171	9.1681	9.4647	9.5684
0.250	6.4724	8.1084	8.8928	9.3531	9.6557	9.7615
0.500	6.5143	8.1616	8.9514	9.4148	9.7194	9.8259

TABLE A3.8

AVERAGE AVAILABILITY AND EXPECTED NUMBER OF FAILURES,
HARDWARE-SOFTWARE SYSTEM

(t = 2000)

AVERAGE AVAILABILITY

SOFTWARE REPAIR RATE	HARDWARE REPAIR RATE 0.020	0.050	0.100	0.200	0.500	1.000
0.010	0.5444978	0.6865691	0.7539209	0.7932200	0.8189707	0.8279540
0.050	0.6281828	0.7930770	0.8710796	0.9165449	0.9463172	0.9566999
0.100	0.6386434	0.8063905	0.8857244	0.9319606	0.9622355	0.9727931
0.250	0.6449197	0.8143786	0.8945113	0.9412100	0.9717865	0.9824490
0.500	0.6470118	0.8170413	0.8974403	0.9442931	0.9749701	0.9856677

EXPECTED NUMBER OF SOFTWARE FAILURES

0.010	11.1111	11.1111	11.1111	11.1111	11.1111	11.1111
0.050	11.1111	11.1111	11.1111	11.1111	11.1111	11.1111
0.100	11.1111	11.1111	11.1111	11.1111	11.1111	11.1111
0.250	11.1111	11.1111	11.1111	11.1111	11.1111	11.1111
0.500	11.1111	11.1111	11.1111	11.1111	11.1111	11.1111

EXPECTED NUMBER OF HARDWARE FAILURES

0.010	10.8926	13.7340	15.0810	15.8670	16.3820	16.5617
0.050	12.5663	15.8642	17.4242	18.3335	18.9290	19.1366
0.100	12.7755	16.1304	17.7171	18.6418	19.2473	19.4585
0.250	12.9010	16.2902	17.8928	18.8268	19.4383	19.6516
0.500	12.9429	16.3434	17.9514	18.8885	19.5020	19.7160

TABLE A3.9

EXPECTED TOTAL COST PER UNIT TIME
HARDWARE-SOFTWARE SYSTEM

$C_{S1}=10$, $C_{H1}=10$, $C_{S2}=10$, $C_{H2}=10$, $C_{S3}=C_{H3}=10$

t = 100

SOFTWARE REPAIR RATE	HARDWARE REPAIR RATE 0.02	0.05	0.10	0.20	0.50	1.00
0.01	9.13	9.40	9.87	10.86	13.85	18.84
0.05	7.07	7.11	7.42	8.29	11.20	16.16
0.10	6.42	6.27	6.46	7.24	10.08	15.02
0.25	6.94	6.59	6.65	7.33	10.10	15.01
0.50	9.07	8.64	8.64	9.28	12.03	16.93

t = 500

0.01	9.14	8.88	9.07	9.88	12.75	17.70
0.05	5.97	4.68	4.39	4.92	7.60	12.49
0.10	5.97	4.55	4.19	4.68	7.34	12.23
0.25	7.17	5.67	5.27	5.74	8.39	13.27
0.50	9.57	8.04	7.63	8.09	10.74	15.61

t = 1000

0.01	9.31	7.88	7.54	8.05	10.72	15.60
0.05	7.06	4.90	4.22	4.53	7.07	11.91
0.10	7.22	4.98	4.26	4.54	7.07	11.90
0.25	8.52	6.22	5.48	5.74	8.26	13.09
0.50	10.96	8.64	7.88	8.15	10.66	15.49

t = 2000

0.01	8.61	6.37	5.66	5.96	8.50	13.34
0.05	7.51	4.86	3.96	4.15	6.62	11.43
0.10	7.82	5.12	4.20	4.37	6.83	11.64
0.25	9.21	6.48	5.54	5.71	8.16	12.97
0.50	11.67	8.93	7.99	8.15	10.60	15.41

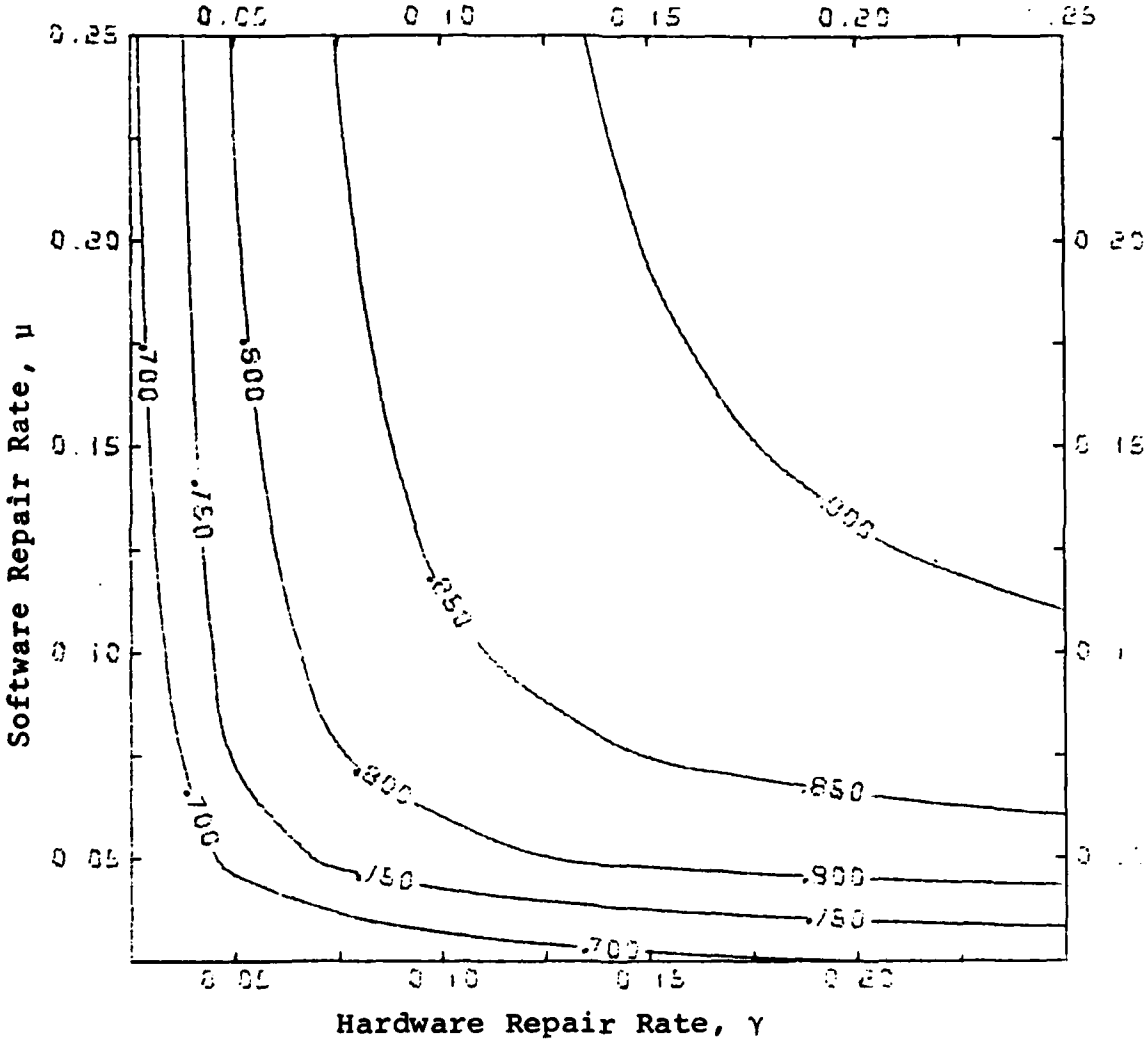


Figure A3.8. Contours of Average Availability vs. Repair Rates: Hardware-Software System (β = .01, λ = .05, t = 500).

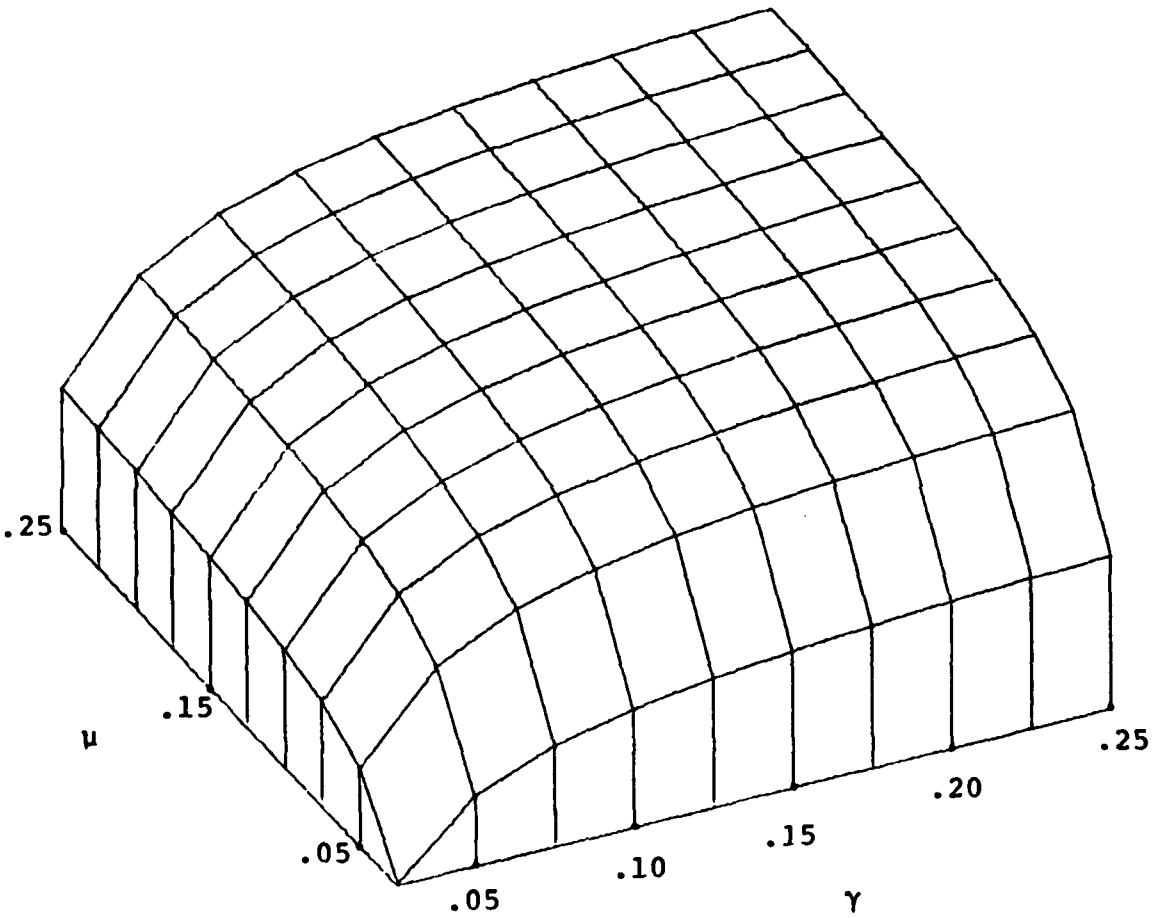


Figure A3.9. Surface of Average Availability vs. Repair Rates: Hardware-Software System (β = .01, λ = .05, t = 500).

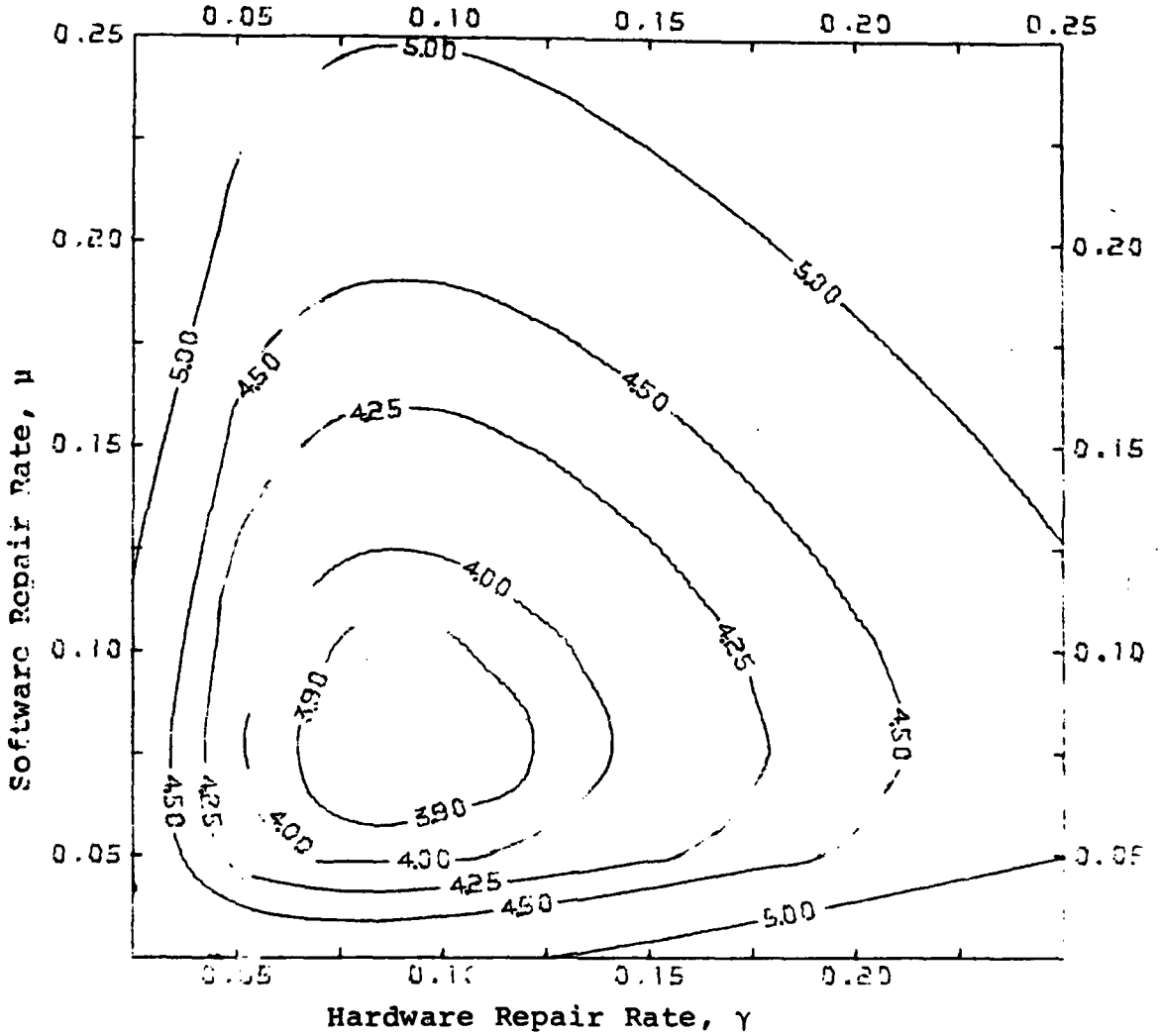


Figure A3.10. Contours of Expected Total Cost/Unit Time vs. Repair Rates: Hardware-Software System ($\beta = .01$, $\lambda = .05$, $t = 500$, cost factors = 10).

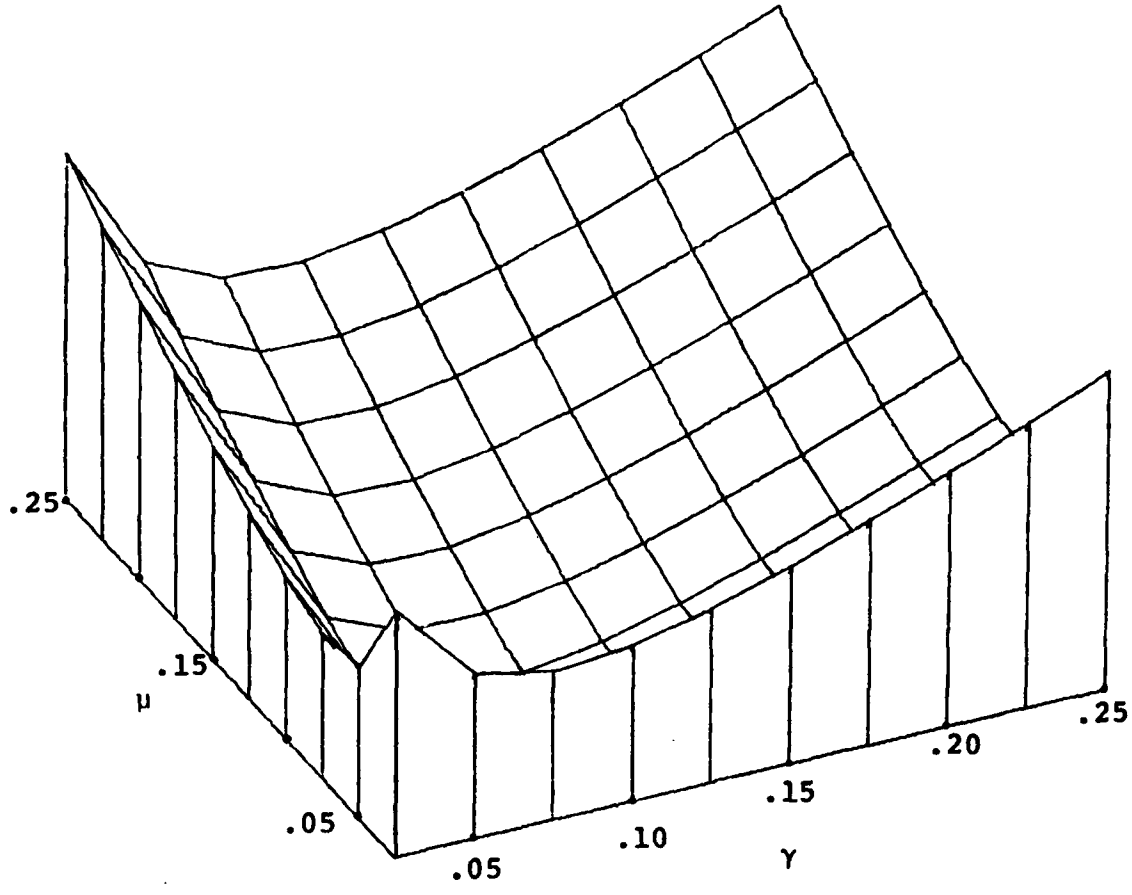
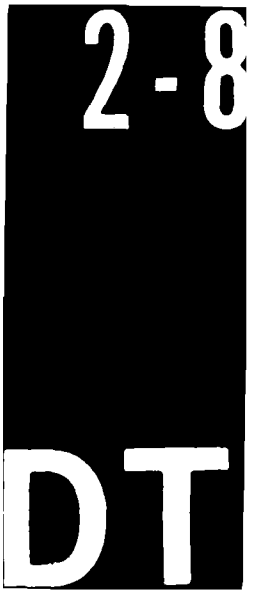


Figure A3.11. Expected Total Cost/Unit Time vs. Repair Rates: Hardware-Software System (β = .01, λ = .05, t = 500).


2-8
DT